Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Tietojärjestelmät

2011

Jaakko Haarala

TIETOJÄRJESTELMIEN YHTEENTOIMIVUUS JA WEB SERVICE -TEKNOLOGIAT

– Kalenteri-integraation toteutus

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

Turun ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma | Tietojärjestelmät

Tammikuu 2011 | 49 + 12

Ohjaaja Tuomo Helo

Jaakko Haarala

TIETOJÄRJESTELMIEN YHTEENTOIMIVUUS JA WEB SERVICE -TEKNOLOGIAT

Yrityksissä on teknologian kehityksen myötä nykyään käytössä yhä enemmän erilaisia toimintoja tehostavia tietojärjestelmiä. Näiden järjestelmien pitää pystyä kommunikoimaan toistensa ja usein myös ulkopuolisten järjestelmien kanssa. Kommunikaation mahdollistamiseksi järjestelmien tulisi olla yhteentoimivia keskenään. Yksi tapa toteuttaa yhteentoimivuuden takaava kommunikaatio on käyttää hyväksi web service -teknologioita.

Työn päätavoitteena on toteuttaa integraatio kalenteritietojen osalta kahden järjestelmän välillä.

Työn teoriaosuudessa selvitetään mitä yhteentoimivuudella tarkoitetaan ja miksi se on tärkeää tietojärjestelmissä. Lisäksi työssä esitellään web service -teknologioita, jotka ovat yksi tapa taata hyvä yhteentoimivuus eri järjestelmien välillä. Työn käytännön osuudessa selvitetään erilaisia lähestymistapoja kalenteri-integraation toteuttamiseen ja miten integraatio toteutettiin käyttämällä epäsuoraa lähestymistapaa.

Työstä tehtyjen päätelmien perusteella web service -teknologiat ovat hyvä vaihtoehto, kun halutaan varmistaa mahdollisimman hyvä yhteentoimivuus järjestelmien välillä. Päätelmissä muistutetaan kuitenkin että web service -implementaatioiden standardituki vaihtelee. Toteutettu integraatiosovellus täytti sille asetetut tavoitteet. Integraation toteutuksen aikana ilmenneet yhteentoimivuusongelmat ovat hyvä esimerkki siitä, minkälaisia ongelmia voi tulla vastaan erilaisten järjestelmien kanssa työskennellessä.

ASIASANAT:

SOAP, web service, web-palvelu, yhteentoimivuus

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Information Systems

January 2011 | 49 + 12

Instructor Tuomo Helo

Jaakko Haarala

INTEROPERABILITY OF INFORMATION SYSTEMS AND WEB SERVICE TECHNOLOGIES

These days modern companies use one or more productivity-enhancing information systems. To maximize productivity, these systems must be able to communicate seamlessly with each other and also with systems outside of the company, and in order to be able to communicate with each other, they have to be interoperable. One way to implement interoperability is to use web service technologies.

The main objective of this thesis was to implement calendar integration between two systems. The theoretical part explains what interoperability means and why it is important in information systems. It also describes what web service technologies are and why they are a potential solution to interoperability problems. The practical part describes different approaches to the calendar integration between the two systems, and how the integration was implemented in the end.

The conclusion was that web service technologies are a good way to ensure interoperability between different systems. However, it also seems that not all web service stacks have the same standard support. Although the calendar integration was implemented successfully in the end, the process serves as an example of what kinds of interoperability problems may arise when working with different systems.

KEYWORDS:

Interoperability, SOAP, web service

SISÄLTÖ

LIITTEET

KUVAT

KUVIOT

TAULUKOT

KÄYTETYT LYHENTEET

API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
MD5	Message-Digest algorithm 5
REST	Representational State Transfer
SDK	Software Development Kit
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
	Service Oriented Architecture Protocol
SSL	Secure Sockets Layer
WCF	Windows Communication Foundation
WSDL	Web Services Description Language (versio 2.0)
	Web Services Definition Language (versio 1.1)
WS-I	Web Services Interoperability Organization
XML	Extensible Markup Language
XSD	XML Schema Definition

1 JOHDANTO

1.1 Tausta

Nykyään erilaiset tietojärjestelmät ovat tärkeä osa nykyaikaisen yrityksen liiketoimintaa. Yrityksissä saattaa olla useampia erikoistuneita järjestelmiä, jotka hoitavat jonkin tietyn osa-alueen yrityksen liiketoiminnassa.

Yrityksen tehokkaan toiminnan kannalta on tärkeää, että järjestelmät voivat jakaa ajan tasalla olevaa tietoa toistensa kanssa. Internetin yleistymisen myötä järjestelmien täytyy usein pystyä myös kommunikoimaan ulkopuolisten järjestelmien, kuten verkkopankkien, kanssa. Järjestelmien välisen kommunikaation toteutuksesta käytetään yleensä termiä integraatio.

Integraation toteutukselle aiheuttavat haasteita eri alustat ja ohjelmointiteknologiat joiden avulla järjestelmät on toteutettu. Integraatio tulisi pyrkiä toteuttamaan teknologialla, joka takaa yhteentoimivuuden erilaisten järjestelmien välillä, järjestelmien alustoista ja niiden toteutusteknologioista riippumatta. Tällöin tietoon on tarvittaessa helppo päästä käsiksi muista järjestelmistä, eikä tieto ole niin sanotusti siilossa josta sitä ei saada helposti käyttöön.

Eräs suosittu ratkaisu ongelmaan on web service -rajapintojen toteutus ja käyttö järjestelmissä.

Opinnäytteen tarkoituksena on tarkastella tietojärjestelmien yhteentoimivuutta yleisellä tasolla ja sitä, miten yhteentoimivuuteen liittyvät haasteet ovat pyritty ratkaisemaan web service -teknologioilla. Lisäksi tarkastellaan, miten web service -teknologioiden avulla voidaan toteuttaa kalenteritietojen integrointi kahden toteutusteknologioiltaan eroavan järjestelmän välillä. Työn tavoitteena on toteuttaa kalenteritietojen integraatio Sofokus iManager- ja Microsoft Exchange Server -järjestelmien välillä.

Työ on luonteeltaan toiminnallinen ja menetelmänä on käytetty case-menetelmää.

1.2 Toimeksiantaja

Opinnäyte on tehty toimeksiantona Sofokus Oy:lle, joka on vaativiin sähköisiin liiketoimintaratkaisuihin erikoistunut yritys. Tarkoituksena on selvittää web service -teknologioiden yhteentoimivuutta ja toteuttaa proof of concept -tyyppinen sovellus, joka mahdollistaa kalenteritietojen synkronoinnin Microsoft Exchange Server- ja Sofokus iManager -järjestelmien välillä.

1.3 Lähtökohdat

Kalenteritietojen integrointia lähdettiin toteuttamaan työtä varten aikaisemmin tehdyn esiselvityksen perusteella. Esiselvityksessä kartoitettiin onko kalenteritietojen integrointi mahdollista ja millä teknologioilla tämä on toteutettavissa.

Esiselvityksen perusteella yhteentoimivuuden ja toteutuksen kannalta varteenotettavin vaihtoehto oli käyttää web service -teknologioita ja hyödyntää Microsoft Exchange Serverin valmiita web service -rajapintoja.

Toteutuksen haasteellisuutta lisää se, että web service -teknologiat eivät ole alustariippumattomuun vuoksi sidottuja mihinkään tiettyyn ohjelmointikieleen tai -teknologiaan. Tästä syystä eri ohjelmointikielillä toteutettujen web service -toteutusten standardituessa on pieniä eroavaisuuksia ja lisäksi ne voivat toimia eri tavalla sellaisissa asioissa, mihin web service -standardissa ei oteta kantaa.

1.4 Rajaus

Työn teoriaosuus on rajattu käsittelemään yhteentoimivuutta ja sen haasteita tietojärjestelmissä yleisellä tasolla. Yhteentoimivuuden lisäksi esitellään web service -teknologioiden tärkeimmät osat sekä asiat, jotka mahdollistavat yhteentoimivuuden niitä käyttäen.

Web service -teknologioiden laajuudesta ja työn tavoitteista takia toiminnallinen osuus on käytettävien toteutusteknologioiden osalta rajattu Microsoft .NET Frameworkin Windows Communication Foundationiin ja Java Metro web service stackiin. Integraation toteutuksen osalta keskitytään esittelemään toteutusvaihtoehtoja ja toteutuksessa vastaan tulleita ongelmia sekä niiden ratkaisuja.

1.5 Työn rakenne

Työssä selvitetään aluksi mitä yhteentoimivuudella tarkoitetaan, erityisesti tietojärjestelmissä. Tämän jälkeen esitellään web service -teknologioita, jotka ovat yksi tunnettu tapa varmistaa mahdollisimman hyvä yhteentoimivuus erilaisten järjestelmien välillä.

Lopuksi tarkastellaan kahden eri järjestelmän, Microsoft Exchange Serverin ja Sofokus iManagerin, välisen kalenteritietojen integraation toteutusta ja siitä saatuja kokemuksia.

2 YHTEENTOIMIVUUS

2.1 Yhteentoimivuuden määritelmä

Informaatioteknologiassa ja ohjelmistoista puhuttaessa yhteentoimivuudella tarkoitetaan yleensä kahden tai useamman järjestelmän mahdollisuutta kommunikoida toistensa kanssa.

Esimerkiksi Institute of Electrical and Electronic Engineers (IEEE) määrittelee termin *interoperability*, eli yhteentoimivuus, seuraavasti:

> "The ability of two or more systems or components to exchange information and to use the information that has been exchanged." (IEEE 1991, 114).

Vapaasti suomennettuna yhteentoimivuudella tarkoitetaan kahden tai useamman järjestelmän tai komponentin mahdollisuutta vaihtaa ja käyttää hyväksi vaihdettua informaatiota.

International Standards Organization (ISO) määrittelee termin vieläkin yksityiskohtaisemmin:

> "The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units." (ISO/IEC 2382-1).

Määritelmää tulkittuna yhteentoimivuus tarkoittaa kykyä kommunikoida, suorittaa ohjelmia ja välittää tietoa eri toiminnallisten yksikköjen välillä sellaisella tavalla, joka vaatii käyttäjältä vain vähän tai ei ollenkaan kyseisten yksikköjen toiminnasta.

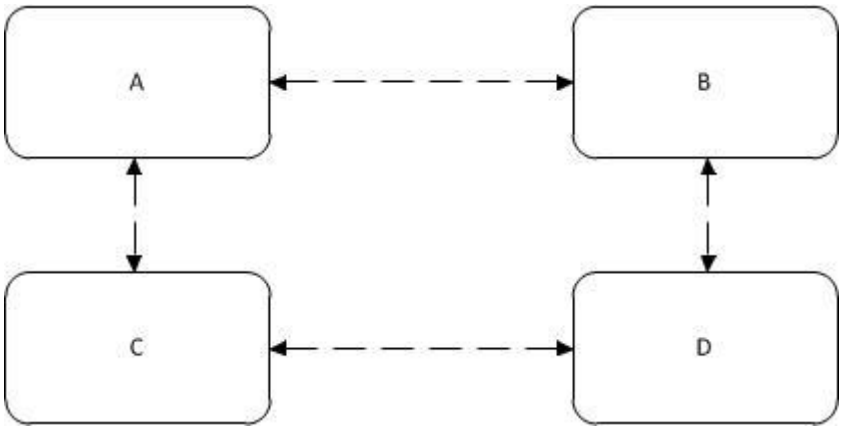
Oleellisin ero IEEE:n määritelmään ISO:n määritelmässä on maininta siitä, ettei käyttäjän tarvitse olla tietoinen toiminnallisten yksikköjen toiminnasta.

Sanastokeskus TSK:n termipankin mukaan yhteentoimivuudella tarkoitetaan tietojärjestelmien kykyä viestiä keskenään sellaisella tavalla tai siinä laajuudessa, että ne voivat rutiinimaisesti käyttää toistensa tuloksia (Sanastokeskus TSK ry 2005).

Yhteentoimivuutta ei tule kuitenkaan sekoittaa toiseen samankaltaiseen termiin, yhteensopivuuteen, vaikka näiden voidaan mieltää liittyvän toisiinsa.

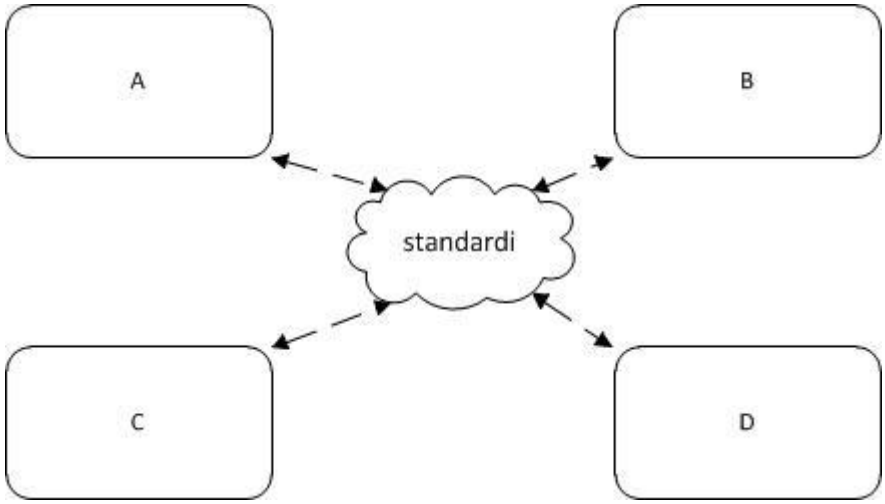
Yhteensopivuudella tarkoitetaan kahden erilaisen järjestelmän mahdollisuutta olla vuorovaikutuksessa toistensa kanssa (IEEE 1991, 45). Sitä voidaan pitää myös yhteentoimivuuden esiasteena.

Kuviossa 1 on havainnollistettu yhteensopivuutta, jossa järjestelmät ovat yhteensopivia vain osan kuvattujen järjestelmien kanssa. Esimerkiksi järjestelmä A on yhteensopiva B:n ja C:n kanssa, muttei D:n kanssa. Toisin sanoen kuvatut järjestelmät eivät ole yhteentoimivia.



Kuvio 1. Yhteensopivat järjestelmät

Yhteentoimivuudella taasen tarkoitetaan sitä, että kaksi tai useampi järjestelmä voivat olla vuorovaikutuksessa toistensa kanssa järjestelmästä riippumatta. Kuviossa 2 järjestelmät A, B, C ja D ovat yhteentoimivia.



Kuvio 2. Yhteentoimivat järjestelmät

Kuviossa 2 kuvattu yhteentoimivuus on saavutettu käyttämällä standardoitua viestintätapaa järjestelmien välisessä kommunikaatiossa.

Yhteentoimivuus rinnastetaan usein suoraan avoimiin standardeihin, koska ne liittyvät vahvasti toisiinsa. Ne olisi kuitenkin hyvä erottaa toisistaan, koska yhteentoimivuus voidaan katsoa saavutettavan myös ilman avoimien standardien käyttöä.

Esimerkiksi jos jokin järjestelmä on saavuttanut niin huomattavan aseman, että sen käyttämää tapaa kommunikoida ulkopuolisten järjestelmien kanssa on tullut niin sanottu de facto -standardi, voidaan olettaa, että altavastaajat haluavat pyrkiä yhteentoimivuuteen kyseisen järjestelmän kanssa implementoimalla tämän tavan kommunikoida järjestelmien välillä.

2.2 Yhteentoimivien järjestelmien hyödyt

Järjestelmien yhteentoimivuudesta on monia hyötyjä. Yksi huomattavimmista hyödyistä on luvussa 1 mainittu tietojen saaminen pois ns. siiloista muiden järjestelmien käytettäväksi.

Koska yhteentoimivat järjestelmät voivat käyttää toistensa tietoja hyväksi suoraan, vähentää se tarvetta manuaaliselle tietojen syötölle järjestelmästä

toiseen. Tällaisen automatisoinnin ansiosta muun muassa manuaalisen työn aiheuttamat virheet ja viiveet vähenevät.

Esimerkiksi yritys jonka asiakkuuden- ja varastonhallintajärjestelmät ovat yhteentoimivia toistensa kanssa pystyy palvelemaan asiakkaitaan tehokkaasti, koska järjestelmät pystyvät kommunikoimaan suoraan toistensa kanssa. Tällöin myyjä voi esimerkiksi nähdä suoraan asiakkuudenhallintajärjestelmästä yrityksen tuotteiden reaaliaikaiset varastosaldot.

Yhteentoimivuuden ansiosta yritys voi lisäksi helposti myös lisätä tai korvata järjestelmiä. Yhteentoimivasta varastonhallintajärjestelmästä voidaan hakea esimerkiksi reaaliaikaiset varastosaldot helposti verkkokauppaan.

2.3 Yhteentoimivuuden haasteet tietojärjestelmissä

Suurimpia haasteita tietojärjestelmien yhteentoimivuuden kannalta ovat käytettävien teknologioiden kirjo. Järjestelmät voivat toimia hyvin erilaisilla alustoilla, minkä lisäksi niiden toteutuksessa käytetyt ohjelmointiteknologiat voivat vaihdella. Lisäksi järjestelmien ominaisuudet ja tietorakenteet voivat erota toisistaan hyvinkin paljon.

Tästä johtuen järjestelmien yhteentoimivuutta varten tarvitaan hyvin tuettu ja alustariippumaton teknologia.

2.4 Yhteentoimivuuden saavuttaminen

Yhteentoimivuuden saavuttamiseksi erilaisten järjestelmien välillä tarkoittaa yleensä sitä, että tarvitaan käyttöön avoin standardi, jonka järjestelmät implementoivat. Tämän avulla järjestelmät voivat teoriassa kommunikoida kaikkien niiden järjestelmien kanssa kyseisen standardin tai teknologian puitteissa.

Yksi tunnettu ja yleisesti käytössä oleva avoimiin standardeihin perustuva toteutus on web service -teknologiat (W3C 2004). Web service teknologioita hyödynnetään usein erilaisissa järjestelmissä kun pyritään mahdollisimman hyvään yhteentoimivuuteen ja alustariippumattomuuteen.

Web serviceen liittyviä standardeja ja spesifikaatioita ylläpitää World Wide Web Consortium (W3C). W3C on kansainvälinen standardiorganisaatio, joka kehittää ja ylläpitää yhdessä sen jäsenten ja muun julkisen yleisön kanssa erilaisia WWW -standardeja (W3C 2010).

3 WEB SERVICE TEKNOLOGIAT

3.1 Määritelmä

Web service -teknologioille ei ole selvitysten perusteella suomen kielessä vakiintunutta käännöstä tai vastinetta, vaan niistä puhuttaessa käytetään yleensä englanninkielistä termiä web service.

Web serviceistä voidaan käyttää suomenkielistä termiä web-palvelu, mutta tällöin se voidaan sekoittaa verkkopalveluihin, joihin eri tahot usein viittaavat web-palveluina.

Sanastokeskus TSK:n tietotekniikan termitalkoiden mukaan suositeltava suomenkielinen termi on www-sovelluspalvelu (Sanastokeskus TSK ry 2009). Termi ei kuitenkaan ole, luultavasti sen tuntemattomuudesta johtuen, yleisesti käytetty.

Tässä opinnäytteessä käytetyllä termeillä web-palvelu ja palvelu, tarkoitetaan nimenomaan web service -teknologioilla toteutettuja palveluita.

Web service -standardi kattaa laajan määrän erilaisia teknologioita. Tästä johtuen termin määritelmä vaihtelee jonkin verran sen mukaan keneltä kysytään. Web service -standardien ja arkkitehtuurin määrittelystä ja ylläpidosta huolehtiva World Wide Web Consortium määrittelee web servicet seuraavasti:

> "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." (W3C 2004).

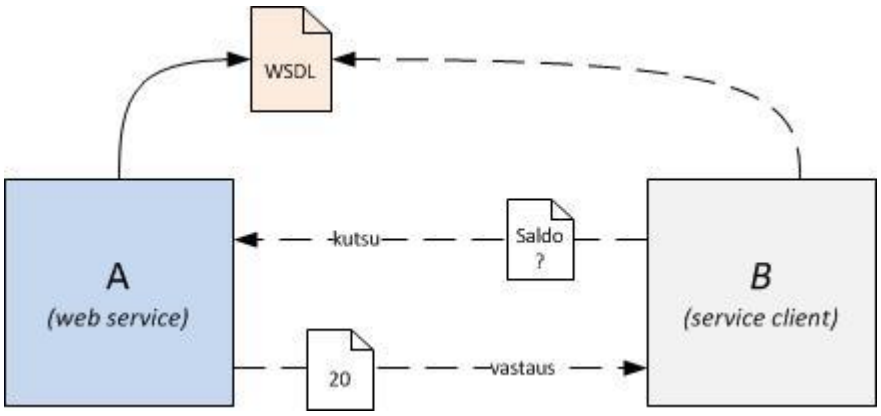
Määritelmää vapaasti tulkittuna web service on siis ohjelmisto, joka tukee ja mahdollistaa eri järjestelmien yhteentoimivuuden ja sen tarjoamien palvelujen käytön verkon yli.

Palvelun käytön mahdollistava rajapinta on määritelty ohjelmallisesti tulkittavissa olevassa formaatissa, josta ilmenee muun muassa miten palvelua käyttävien järjestelmien tulee kommunikoida palvelun kanssa.

Muut järjestelmät kommunikoivat palvelun kanssa käyttämällä XML:n pohjautuvilla, niin sanotuilla SOAP-viesteillä. Useimmiten kommunikaatio tapahtuu HTTP-protokollan välityksellä.

Kyseessä on siis eräänlainen verkossa sijaitseva ja verkon kautta käytettävä sovellus, joka tarjoaa erilaisia toimintoja eli palveluita muiden käytettäväksi (Kalin 2009, 1).

Pohjimmiltaan voidaan sanoa kyseessä olevan eräänlainen palvelun tarjoajan ja asiakkaan välinen suhde, jossa asiakas käyttää hyväksi tarjottuja palveluja. Kuviossa 3 on yksinkertaistettuna web service -palvelujen toimintaidea.



Kuvio 3. Web service -esimerkki

Kuviossa 3 on järjestelmä A, joka tarjoaa web service -rajapinnan kautta tietoa varastosaldosta. Järjestelmän kanssa kommunikoimiseen vaadittavat tiedot ovat kuvattu tämän julkaisemassa WSDL-dokumentissa.

Järjestelmä B toimii A:n asiakkaana, joka pyytää järjestelmältä A tietoja varaston saldosta. Jotta B voi onnistuneesti kommunikoida A:n kanssa, se hakee ensin A:n julkaisemasta palvelumääritelmästä (WSDL-dokumentti) tarvittavat tiedot A:n kanssa kommunikoimiseen. Näiden tietojen pohjalta B rakentaa viestin, jonka avulla voidaan pyytää A:lta saldotietoja.

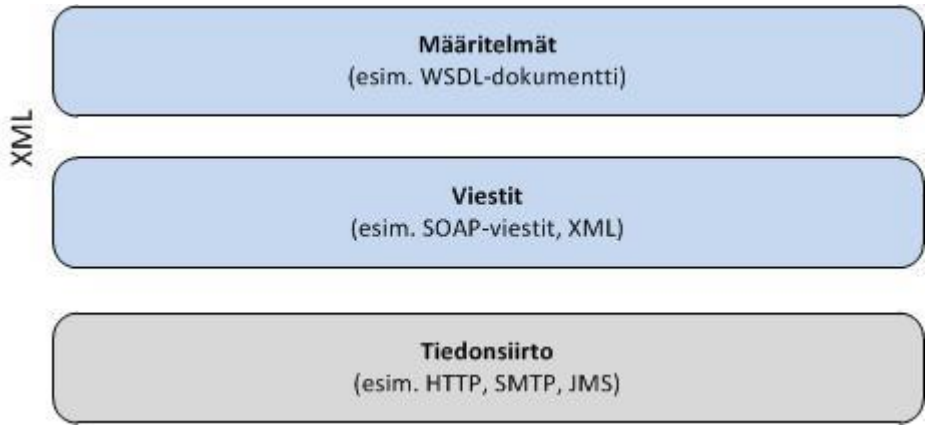
Kutsun vastaanottamisen jälkeen järjestelmä A suorittaa tarvittavat toimenpiteet ja lähettää se palvelumääritelmässä (WSDL-dokumentti) määritellyn vastausviestin järjestelmälle B.

3.2 Käytetyt teknologiat ja rakenne

3.2.1 Rakenne

Web servicet rakentuvat pitkälti tunnettujen ja hyvin tuettujen teknologioiden varaan. Tiedonsiirrossa voidaan hyödyntää esimerkiksi HTTP-protokollaa ja välitetyt viestit ovat usein XML-pohjaisia. (Kalin 2009, 1)

Kuviossa 4 on yksi tapa kuvailla web servicen perusrakennetta.



Kuvio 4. Web servicen perusrakenne

Viestien välitykseen voidaan käyttää HTTP-protokollaa tai muuta käyttötarkoitukseen sopivaa tiedonsiirtotapaa. Välitettävät viestit, palvelumääritelmät ja muu informaatio taas perustuu XML-kieleen (W3C 2004).

W3C:n web service -määritelmässä SOAP-viestejä ja WSDL-dokumentteja pidetään tärkeänä osana web servicejä. Web service voidaan toteuttaa kuitenkin myös ilman kyseisiä teknologioita, esimerkiksi käyttämällä puhdasta XML:ää.

Yhteentoimivuuden kannalta tämä on kuitenkin epäedullisempi tapa, koska ei voida olla täysin varmoja viestien sisällöistä ja palvelun käytöstä, koska ne

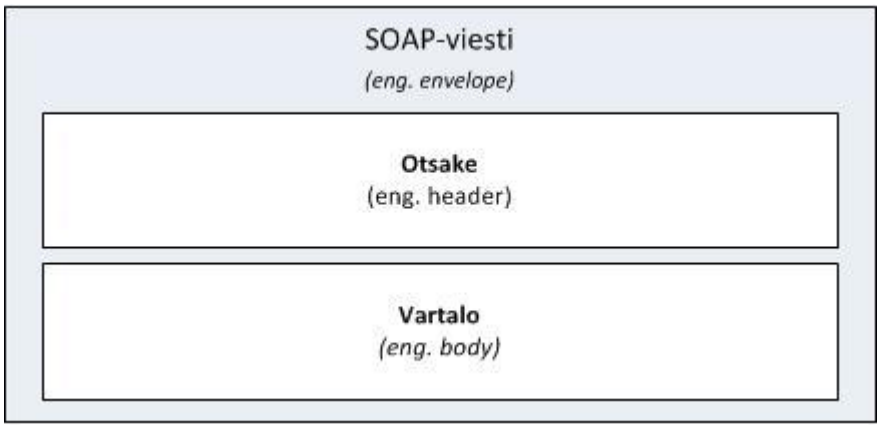
voivat olla löyhemmin määriteltyjä, eivätkä perustu välttämättä mihinkään tiettyyn standardiin.

3.2.2 SOAP

SOAP on XML:ään perustuva alustariippumaton tietoliikenneprotokolla. Pohjimmiltaan SOAP:n perusideana on mahdollistaa rakenteellisen tiedon välittäminen erityisesti hajautettuja järjestelmiä ajatellen. SOAP:n suunnittelun päätavoitteita ovat olleet yksinkertaisuus ja helppo laajennettavuus. (W3C 2007.)

SOAP on ollut alun perin lyhenne sanoista Simple Object Access Protocol, mutta viimeisimmän version 1.2 myötä tästä käytännöstä on luovuttu (W3C 2007).

SOAP-viestin rakennetta voidaan kuvailla eräänlaiseksi kirjekuoreksi. W3C:n SOAP-spesifikaatiossa lähetettävää viestikokonaisuutta kutsutaankin SOAP-envelopeksi, eli SOAP -kirjekuoreksi, joka sisältää muun muassa varsinaisen viestiosan (W3C 2007). Kuviossa 5 on kuvattu SOAP-viestin rakennetta.



Kuvio 5. SOAP-viestin rakenne

SOAP-viesti koostuu kahdesta elementistä, jotka ovat spesifikaatiossa nimetty headeriksi ja bodyksi, vapaasti suomennettuna otsakkeeksi ja vartaloksi. Näistä kahdesta header -elementti on valinnainen eli viestin ei tarvitse sisältää sitä, kun taas body -elementti on pakollinen osa SOAP-viestiä. (W3C 2007.)

Header -elementti mahdollistaa SOAP-viestien laajentamisen. Käytännössä tämä tarkoittaa sitä, että siihen voidaan sisällyttää sovelluskohtaista tietoa, kunhan se tehdään SOAP-standardissa määritellyllä tavalla. (W3C, 2007.)

Header -elementtiin lisättäviin tietoihin voidaan lisäksi määritellä, pitääkö vastaanottajan ymmärtää lisätty tieto. Vastaanottajan tulee käsitellä ymmärrettäväksi merkitty tieto, mutta vapaaehtoiseksi merkitty tieto voidaan jättää käsittelemättä mikäli vastaanottaja ei osaa sitä käsitellä.

Web service -teknologioilla toteutetut palvelut voivat esimerkiksi sisällyttää SOAP-viestien header-elementtiin käyttäjien autentikointitietoja tai tietoja viestin salauksesta.

Body-elementti sisältää viestin varsinaisen sisällön.

Seuraavassa on esimerkki siitä, miltä yksinkertainen SOAP-viesti voi käytännössä näyttää:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:HaeVarastoSaldo xmlns:m="http://palvelu/varastosaldo">
      <m:Tuote>polkupyörä</m:Tuote>
    </m:HaeVarastoSaldo>
  </soap:Body>
</soap:Envelope>
```

Esimerkin SOAP-viestin voidaan havaita koostuvan vapaaehtoisesta header-elementistä ja pakollisesta body-elementistä. Body-elementti eli varsinainen viestiosa sisältää lapsielementin josta käy ilmi, että lähettäjä yrittää hakea varastosaldoa tuotteelle polkupyörä.

3.2.3 WSDL

WSDL (Web Services Definition Language tai Web Services Description Language) on XML-pohjainen kuvauskieli jota käytetään web service -teknologioilla toteutettujen palvelujen kuvailemiseen.

WSDL-dokumentti sisältää tiedon palvelun tarjoamista operaatioista, mitä parametreja nämä ottavat vastaan ja sen käyttämistä tyypeistä. Lisäksi se sisältää tiedon operaatioiden palautusarvoista ja palvelun endpointista tai portista. (Kalin 2009 17)

WSDL-dokumenttia kutsutaan usein myös *service contractiksi* eli palvelu-sopimukseksi, koska siinä on määritelty miten dokumentin kuvaileman palvelun ja sen rajapintojen kanssa tulee kommunikoida (Resnick ym. 2008, 37).

WSDL-standardista on olemassa kaksi versiota, versiot 1.1 ja 2.0. Muutoksia versioiden välillä ovat muun muassa PortTypes- ja Ports-elementtien uudelleen nimeäminen sekä Message-elementin poistamien (Dhesiaseelan 2004).

WSDL-dokumentti koostuu versiosta riippuen 6-7 elementistä, jotka on listattu taulukossa 1.

Taulukko 1. WSDL-dokumentin elementit versioittain

Version 1.1 elementit	Version 2.0 elementit
Types	Types
Message	-
Operation	Operation
Port Type	Interface
Binding	Binding
Port	Endpoint
Service	Service

Types -elementti sisältää palvelun käyttämien tietotyyppien määritelmät. Tyypit on määritelty käyttämällä jotain tiettyä tietotyyppien määrittelytapaa (Data Type System) (Kalin 2009, 37).

Oletuksena tietotyyppien määrittelyssä WSDL-spesifikaatiossa suositaan XML Schema Definitionin (XSD) käyttöä parhaan yhteentoimivuuden ja alustariippumattomuuden takaamiseksi (W3C 2001).

Message-elementissä määritellään palvelun kanssa kommunikointiin käytettävien viestien rakenne (Kalin 2009, 37). Kuten taulukosta 1 käy ilmi, WSDL:n versiossa 2.0 Message-elementin käytöstä on luovuttu.

Operation-elementti määrittelee yksittäisen operaation jonka palvelu voi suorittaa. Operation-elementti tai elementit määritellään PortType- tai Interface-elementissä eli se on näiden lapsielementti. (W3C 2001.)

PortType- tai Interface-elementissä määritellään palvelun tarjoamat operaatiot sekä viestit, jotka liittyvät tietyn operaation käyttämiseen.

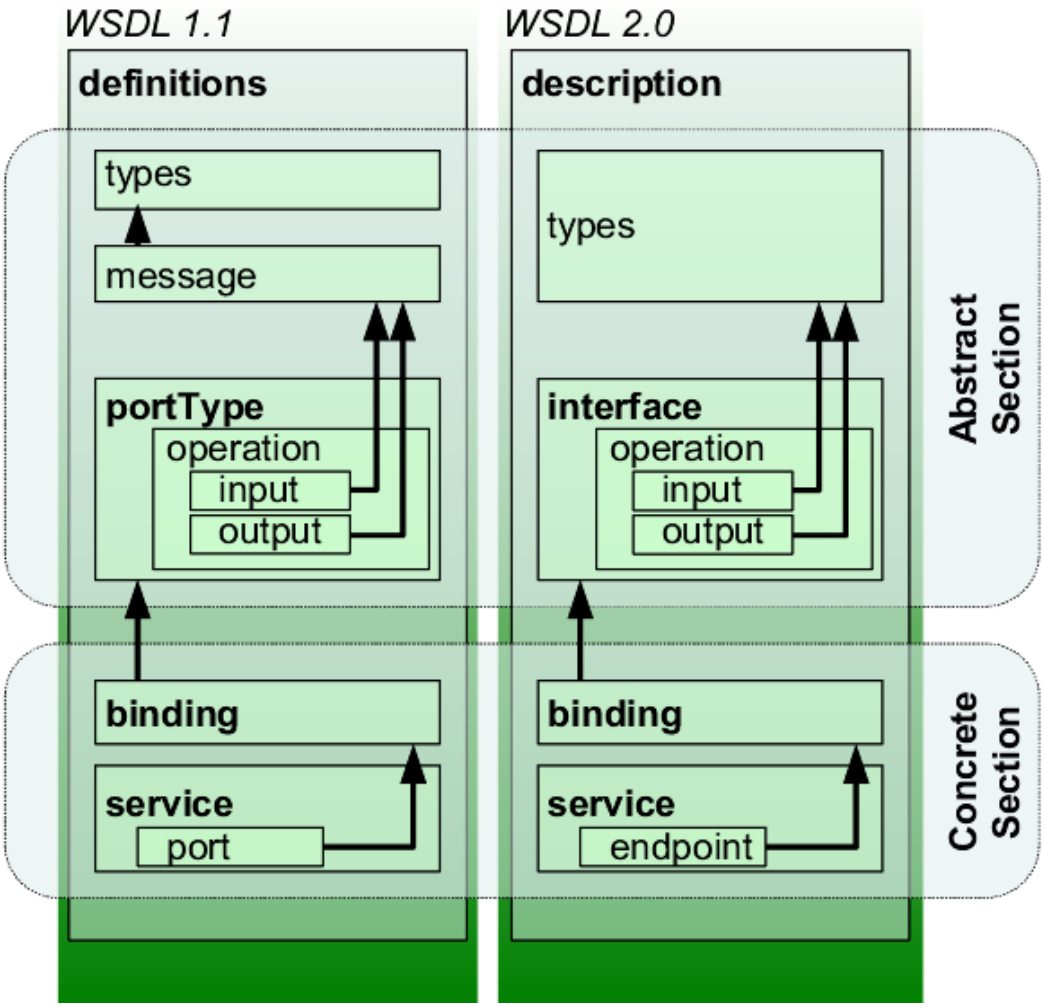
Binding-elementti määrittelee viestiformaatin ja protokollat tietyssä portType- tai Interface-elementissä määritellyille operaatioille ja viesteille (W3C 2001). Sen voidaan sanoa toimivan sidoksena kuviossa 6 merkittyjen WSDL-dokumentin abstraktin ja konkreettisen osion välillä.

Port- tai Endpoint-elementti on olennainen osa WSDL-dokumenttia ja se määrittelee yhden päätepisteen eli osoitteen, jonka kautta palvelun kanssa voidaan kommunikoida (W3C 2001). Port- tai endpoint-elementit ovat service-elementin lapsielementtejä.

Service-elementti kokoaa yhteen palvelun port- tai endpoint-määritykset, joiden kautta palvelun toimintoja käytetään. Service-elementissä on mahdollista määritellä useampia portteja tai endpointteja. (W3C 2001.)

Liite 1 sisältää esimerkin yksinkertaisesta WSDL:n version 1.1 mukaisesta WSDL-dokumentista, josta käy ilmi elementtien suhteet toisiinsa.

WSDL-dokumentin rakennetta on havainnollistettu kuviossa 6.



Kuvio 6. WSDL-dokumentin rakenne versioissa 1.1 ja 2.0 (Wikipedia 2010).

Kuviosta 6 selviää taulukossa 1 listattujen elementtien suhteet toisiinsa WSDL:n versioissa 1.1 ja 2.0. Lisäksi kuvio havainnollistaa versioiden välisiä eroja, kuten esimerkiksi message-elementin puuttuminen versiosta 2.0 ja sen korvaaminen kokonaan types-elementillä.

Kuvioon merkitty konkreettinen osio koostuu yhdestä tai useammasta service -elementistä. Konkreettiseksi elementit tekee niille määritelty port- tai endpoint -elementti joka sisältää palvelun osoitteen.

Kuvion abstraktiksi merkitty osio koostuu abstrakteista asioista, jotka liittyvät palvelun tarjoamiin operaatioihin ja niiden käyttöön tarvittaviin viesteihin.

WSDL:n eri versioiden tuki vaihtelee eri toteutusten ja ohjelmointiteknologioiden välillä. Esimerkiksi .NET framework 4:n Windows Communication Foundation tukee vain WSDL:n versiota 1.1 (Microsoft Corporation 2010d).

3.3 Jaottelu alakategorioihin

3.3.1 Jakoperusteet

Web service -teknologioilla toteutetut palvelut voidaan jakaa karkeasti kahteen kategoriaan, REST-tyylisiin- ja SOAP-pohjaisiin palveluihin niiden toteutuksen mukaan (Kalin 2009, 1).

3.3.2 REST-tyyliset palvelut

REST-tyyliset palvelut ovat suosittuja muun muassa kuluttajille suunnatuissa palveluissa. Sen etuja ovat keveys ja yksinkertaisuus (Resnick ym. 2008, 2). Esimerkkinä REST-palvelujen yksinkertaisuudesta on se, että viestien formaattina ei ole pakko käyttää SOAP-standardissa määriteltyjä viestejä, vaan ne voidaan tarvittaessa määritellä vapaasti itse. Lisäksi palvelun kuvaukseen ei ole myöskään pakko käyttää WSDL-dokumenttia.

Esimerkiksi tunnettu mikrobloggauspalvelu Twitter tarjoaa tavanomaisen internetsivuston lisäksi REST-tyylisen web service -rajapinnan, joka mahdollistaa palvelun käytön osana muita sovelluksia (Twitter Inc. 2010).

3.3.3 SOAP-pohjaiset palvelut

SOAP-pohjaiset palvelut ovat taasen suosittuja yritysmaailman järjestelmissä tiukemman määrittelyn ansiosta (Resnick ym. 2008, 2). Viestit ovat tällöin SOAP-viestejä ja tämän lisäksi palveluista on usein saatavilla myös WSDL-dokumentti.

Tarkemman määrittelyn ja hyvän ohjelmointityökalujen tuen takia SOAP-viesteihin perustuvat web servicet ovat suosiossa erityisesti yritysmaailman järjestelmissä. Muun muassa Microsoftin Exchange Server 2007 ja SharePoint

2007 -järjestelmissä on toteutettu web service-rajapinnat, jotka mahdollistavat niiden kautta käytettävissä olevien toimintojen kutsumisen järjestelmien ulkopuolelta.

Esimerkkinä SOAP-pohjaisesta palvelusta, jossa on pyritty mahdollisimman hyvään yhteentoimivuuteen erilaisten järjestelmien välillä, on suomalaisten pankkien tarjoama Tupas-varmennepalvelu, jonka avulla verkkopalvelut, kuten esimerkiksi verkkokaupat, voivat tunnistaa asiakkaansa (Finanssialan Keskusliitto 2010).

3.4 Palveluiden tietoturva

3.4.1 Yleistä

Koska tiedonsiirto tapahtuu verkon yli, on tietoturvaan kiinnitettävä erityistä huomiota, ettei tiedonsiirtoa web-palvelun kanssa voida esimerkiksi salakuunnella, viestien sisältöä muokata tai tehdä muita toimenpiteitä, jotka vaarantavat tietoturvallisuuden.

Web service -pohjaisten palveluiden tietoturvan takaamiseen käytetyt menetelmät voidaan jaotella kahteen erilaiseen toteutustapaan, tiedonsiirtopohjaiseen (Transfer-based security) ja viestipohjaiseen (Message-based security). (Resnick ym. 2008, 317-318)

Tiedonsiirtopohjaisessa toteutustavassa palvelun ja palvelua käyttävän asiakkaan kanssakäyminen pyritään turvaamaan käyttämällä tiedonsiirtotavan tarjoamia tietoturvaominaisuuksia. Viestipohjaisessa toteutustavassa tietoturvatoimenpiteet toteutetaan viestitasolla.

Tiedonsiirto- ja viestipohjaiset tavat eivät näin ollen ole toisiaan poissulkevia tapoja, vaan niiden tarjoamia eri tietoturvaominaisuuksia on mahdollista käyttää yhdessä.

Viestipohjainen tapa on ominaisuuksiltaan tiedonsiirtopohjaista tapaa laajempi ja näin ollen monimutkaisempi, koska autentikoinnin ja viestiliikenteen

salaamisen lisäksi sillä voidaan toteuttaa muita tietoturvallisuuteen liittyviä toimenpiteitä. Tällaisia toimenpiteitä ovat esimerkiksi yhtenäisyyden varmistaminen siltä varalta, että tietoa on muutettu ennen kuin se saapuu lähettäjältä vastaanottajalle (Resnick ym. 2008, 317).

3.4.2 Tiedonsiirtopohjainen suojaustapa

Tiedonsiirtopohjaisessa suojaustavassa suojataan palvelun ja asiakkaan välinen tiedonsiirto käyttämällä esimerkiksi HTTPS-yhteyttä, jossa liikenne on salattu käyttämällä SSL-salausta.

Käyttäjien autentikoimiseen tiedonsiirtopohjaisessa tavassa voidaan käyttää hyväksi HTTP-protokollaan kuuluvia basic- ja digest-autentikointitapoja.

Basic-autentikoinnissa käyttäjän kirjautumistietoja ei salata millään tavalla, minkä takia sen käyttöä tulisi välttää salaamattomilla yhteyksillä (The Internet Engineering Task Force 1996, 48).

Basic-autentikointia turvallisempi tapa tunnistaa käyttäjä on digest-autentikoinnin käyttö. Digest-autentikoinnissa kirjautumistiedot ovat pyritty suojaamaan käyttämällä MD5-pohjaista salausta (IETF 1997, 3).

Tiedonsiirtopohjainen suojaustapa on yhteentoimivuuden kannalta hyvä, koska se ei ole millään lailla riippuvainen siirrettävistä viesteistä tai niiden rakenteista. Tämän lisäksi usein tiedonsiirrossa käytettävä HTTP-protokolla ja sen eri ominaisuudet ovat alustasta riippumatta hyvin tuettuja, minkä vuoksi sen ominaisuuksia voidaan hyödyntää tehokkaasti ilman yhteentoimivuusongelmia.

3.4.3 Viestipohjainen suojaustapa

Viestipohjainen suojaustapa on tiedonsiirtopohjaiseen tapaan verrattuna web service -pohjaisten palvelujen kannalta laajempi, mutta samalla myös monimutkaisempi.

Viestipohjaiseen suojaustapaan käytettävät eri menetelmät ovat kokoelma spesifikaatioita, jotka ovat osa WS-Security -spesifikaatiota. WS-Security on

OASIS:n (Organization for the Advancement of Structured Information Standards) ylläpitämä standardi, joka kuvaa erilaisia tapoja suojata SOAP-pohjaisten palveluiden viestit. (IBM Corporation 2004.)

Viestipohjaisen suojaustavan toiminta perustuu SOAP-standardissa määriteltyihin laajennusmahdollisuuksiin. Viestissä käytetyt tietoturvaominaisuudet määritellään SOAP-viestin header -elementissä. Käytettyjen tietoturvaominaisuuksien mukaan vastaanottaja pystyy kyseisten ominaisuuksien header-elementissä olevien määritelmien avulla tulkitsemaan esimerkiksi, mitä salausalgoritmia body-elementin sisällön salaamiseen on käytetty, tai varmistamaan viestin alkuperän. (Microsoft Corporation 2002.)

3.5 Yhteentoimivuus

Web servicet ovat yhteentoimivuuden kannalta hyvä vaihtoehto, koska ne ovat avoimia ja perustuvat tunnettuihin sekä hyvin tuettuihin teknologioihin. Lisäksi web service -standardin avoimuus mahdollistaa sen, että kuka tahansa voi toteuttaa sen pohjalta web service -pohjaisten palveluiden toteuttamista varten vaadittavat ohjelmistokomponentit.

Yhteentoimivuutta ei voida kuitenkaan taata kokonaan, koska web serviceihin liittyviä teknologioita on mahdollista laajentaa. Esimerkiksi luvussa 3.2.2 kuvattuihin SOAP-viesteihin voidaan sisällyttää mukaan sellaisia laajennoksia, jotka voivat aiheuttaa yhteentoimivuusongelmia, mikäli vastapuoli ei osaa tulkita laajennosta tai jättää sen kokonaan tulkitsematta.

Standardien määrä on suuri ja niiden eri osa-alueita ylläpitävät useampi kuin yksi organisaatio. Lisäksi oman ongelmansa tuottavat joidenkin standardien päällekkäisyydet, mikä aiheuttaa standardien implementoijien kannalta ongelmia. (Kalin 2009, 286)

Yhteentoimivuusongelmien välttämiseksi on olemassa web service -teknologioiden parissa toimivista tahoista koostuva organisaatio Web Services Interoperability Organization (WS-I). WS-I:n päämääränä on kehittää hyviä käytäntöjä liittyen web service -teknologioiden implementoimiseen eri alustoilla.

Näitä hyviä käytäntöjä kutsutaan yhteentoimivuus profiileiksi, jotka koostuvat eri web service -teknologioihin liittyvistä standardeista. (Web Services Interoperability Organization, 2009.)

3.6 Käyttökohteet ja hyödyt

Koska web service -teknologiat eivät ole sidottu mihinkään tiettyyn ohjelmointikieleen tai teknologiaan, mahdollistaa ne eri teknologioilla toteutettujen järjestelmien yhteentoimimisen. Erityistä hyötyä tästä on esimerkiksi hajautetuissa järjestelmissä ja erilaisissa internetin kautta käytettävissä palveluissa.

Web service-teknologiat ovat usein tärkeässä roolissa SOA-arkkitehtuurissa eli palvelukeskeisessä arkkitehtuurissa. Palvelukeskeisessä arkkitehtuurissa tietojärjestelmän eri toiminnot ja prosessit ovat itsenäisiä toisistaan riippumattomia palveluita, joiden käyttö ei ole rajoitettu tiettyihin alustoihin tai teknologioihin. Kyseiset palvelut voivat olla myös hajautettuna useammalle alustalle. Alustariippumattomana web service-teknologiat tarjoavat toimivan ratkaisun kyseiseen ongelma-alueeseen SOA-arkkitehtuurissa. (Colan 2004.)

4 SOAP-POHJAISET PALVELUT KÄYTÄNNÖSSÄ

4.1 Toteutustavat

Web servicet voidaan jaotella arkkitehtuuriin lisäksi myös niiden toteutustavan perusteella kahteen eri kategoriaan, koodikeskeiseen ja dokumenttikeskeiseen lähestymistapaan. Näitä tapoja voidaan myös tietyin edellytyksin käyttää yhdessä. (Kalin 2009, 69-70.)

Koodikeskeisessä lähestymistavassa web servicen rajapinta ja toiminta määritellään suoraan sovelluksen lähdekoodissa. Ajon aikana sovellus generoi automaattisesti tarvittavat palvelumäärittelyt (WSDL-dokumentti), joita hyödyntämällä asiakassovellukset voivat kommunikoida palvelun kanssa.

Dokumenttikeskeisessä lähestymistavassa luodaan ensin palvelun palvelumäärittely eli WSDL-dokumentti. Kun määrittelydokumentti on tehty, sen avulla voidaan generoida tarvittavat lähdekoodit web service -pohjaisen palvelun toiminnallisuuden toteuttamiseen. Yleensä tällöin puhutaan ns. proxy-luokista.

Lähdekoodin generoimista varten WSDL-dokumenteista on olemassa useille eri alustoille. Esimerkiksi Microsoftin .NET Frameworkille on olemassa oma työkalunsa lähdekoodin generoimiselle, samoin kuin Javan Metrolle.

Työkalut pystyvät generoimaan WSDL-dokumentin pohjalta palvelujen toteuttamista varten tarvittavien proxy-luokkien lisäksi asiakasohjelmaan tarvittavat lähdekoodit, koska nämä käyttävät suurelta osin samoja tietoja. Erot ovat lähinnä siinä, että palvelun ja asiakasohjelman toteutukselle luodaan omat luokkansa. Useimmissa työkaluissa voidaan päättää luodaanko pelkästään palvelua tai asiakasohjelmaa varten tarvittavat luokat.

4.2 Toteutustapojen hyvät ja huonot puolet

Koodi- ja dokumenttikeskeisissä lähestymistavoissa on molemmissa omat hyvät ja huonot puolensa.

Koodikeskeisen lähestymistavan huonoihin puoliin kuuluvat se, että lähdekoodiin tehtävät muutokset voivat aiheuttaa muutoksia ajonaikana generoitaviin palvelumäärittelyihin (Kalin 2009, 69). Nämä muutokset voivat aiheuttaa sen, että asiakasohjelmat jotka yrittävät kommunikoida käyttämällä aiempaa palvelumääritystä eivät voi enää kommunikoida palvelun kanssa.

Dokumenttikeskeisen lähestymistavan suurin ongelma on dokumenttien aikaa vievä määrittely ja ylläpito (Kalin 2009, 70). Suurissa järjestelmissä WSDL-dokumentista ja siihen liittyvistä muista dokumenteista voi tulla hyvin pitkiä ja tästä syystä vaikeasti ylläpidettäviä. Dokumenttikeskeisen lähestymistavan etuja on kuitenkin se, että voidaan olla täysin varmoja web service -rajapinnan pysymisestä samana.

4.3 Toteutusteknologiat

Web service -pohjaisten palveluiden toteuttamiseen on olemassa tarvittava tuki useilla eri ohjelmointikielillä ja alustoilla. Suosituimpia ohjelmointiteknologioita web service -pohjaisten palveluiden toteuttamisella ovat Microsoftin .NET framework ja Oraclen Java.

4.3.1 .NET Framework

.NET frameworkissa web service -pohjaiset palvelujen toteuttaminen on mahdollista käyttämällä siihen kuuluvaa Windows Communication Foundationia. WCF on ollut osa .NET frameworkia sen versiosta 3 lähtien, ja se on nykyään suositeltu .NET -teknologia käytettäväksi palveluperustaisten sovellusten toteutuksessa.

Toisin kuin aikaisempi tapa, ASP.NET Web Services, toteuttaa web service -pohjaisia palveluja, WCF mahdollistaa useampia erilaisia tapoja palvelujen julkaisemiseen. Siinä missä ASP.NET -pohjaiset palvelut vaativat käytännössä IIS-palvelinohjelmiston, WCF:n perustuvat palvelut voidaan toteuttaa asp.net -sivujen lisäksi täysin itsenäisinä "stand alone" -sovelluksina tai Windows Serviceinä.

4.3.2 Java

Web service -pohjaisten palveluiden toteuttamiseen Java-kielellä on olemassa monia eri vaihtoehtoja. Tunnettuja Java-pohjaisia web service -toteutusteknologioita ovat Apache Axis, Apache CFX sekä Metro. Näistä kolmesta Metro on Oraclen ylläpitämä ja sitä pidetään eräänlaisena referenssitoteutuksena.

Sovelluskehittimissä Metro on hyvin tuettu muun muassa NetBeans IDE:ssä. NetBeansin hyvän tuen lisäksi Eclipselle on saatavilla hyvin erilaisia laajennoksia Metroon perustuvien palvelujen toteuttamiseksi.

4.3.3 Metron ja WCF:n yhteentoimivuus

Metron kehityksessä on panostettu erityisesti yhteentoimivuuteen WCF:n kanssa ja yhteentoimivuutta testataan tietyin väliajoin Microsoftin ja Oraclen toimesta. Testit perustuvat WS-I:n suosituksiin soveltuvilta osin. (Microsoft Corporation 2010c.)

Tästä syystä Metro on hyvä vaihtoehto, kun halutaan varmistua hyvästä yhteentoimivuudesta WCF-pohjaisten palvelujen kanssa.

4.4 Esimerkki koodikeskeisestä lähestymistavasta

Koodikeskeinen lähestymistapa palveluiden toteutuksessa ei toteutusteknologiasta riippuen juurikaan eroa muusta ohjelmoinnista, jossa määritellään rajapintoja (interface) ja luokkia (class).

Esimerkissä toteutetaan luvussa 3.1 esimerkkinä käytettyä varastosaldo palvelua. Toteutusteknologiana on käytetty .NET frameworkin version 4 Windows Communication Foundationia.

4.4.1 Palvelun toteutus

Esimerkissä varasto-palvelu tarjoaa käyttäjille kahta toimintoa, tuotteiden varastosaldojen hakua ja myyntitapahtumien kirjaamista.

Palvelun tarjoamat toiminnot ovat määritelty ohjelmassa oheisessa interface-luokassa:

```
[ServiceContract(Namespace="http://VarastoEsimerkki/")]
interface IVarasto
{
    [OperationContract]
    int getSaldo(string tuote);

    [OperationContract]
    void myy(string tuote, int maara);
}
```

Tästä voidaan havaita kuinka WCF-pohjaisen palvelun rajapinnan määrittely eroaa tavanomaisesta ohjelmoinnista. IVarasto -interface on merkattu ServiceContract -annotaatiolla, jonka avulla ohjelman käännösvaiheessa tunnistetaan, että se kuvaa web service -rajapintaa. Tämän lisäksi palvelun kautta tarjottavat metodit eli toiminnot on merkitty OperationContract -annotaatioilla.

Palvelun varsinainen toiminnallisuus toteutetaan erillisessä luokassa, implementaatiossa, joka periytyy IVarasto-interfacesta. Palvelun implementoivassa luokassa voidaan niin ikään käyttää erilaisia annotaatioita, kuten seuraavasta esimerkistä käy ilmi:

```
    [ServiceBehavior(InstanceContextMode =
InstanceContextMode.Single, Name = "VarastoService", Namespace =
"http://VarastoEsimerkki/")]
    class VarastoService : IVarasto
    { ...
```

Palvelun sisäistä toimintaa on esimerkissä määritelty ServiceBehavior -annotaatiolla ja se sisältää muun muassa määrittelyn palvelimen ulospäin näkyvästä nimestä.

Oletuksena WCF -pohjaiset palvelut luovat jokaiselle palvelun vastaanottamalle kyselylle oman instanssin. Varastosaldon muutoksien havainnollistamista varten oletustoiminnallisuutta on vaihdettu esimerkissä siten, että jokaista kyselyä palvelee sama instanssi.

Jotta palvelu voidaan käyttää, se pitää julkaista. Esimerkin varastopalvelu on toteutettu niin sanottuna stand alone -konsolisovelluksena, jolloin se ei tarvitse toimiakseen esimerkiksi IIS-palvelinohjelmistoa, vaan se toimii täysin itsenäisesti.

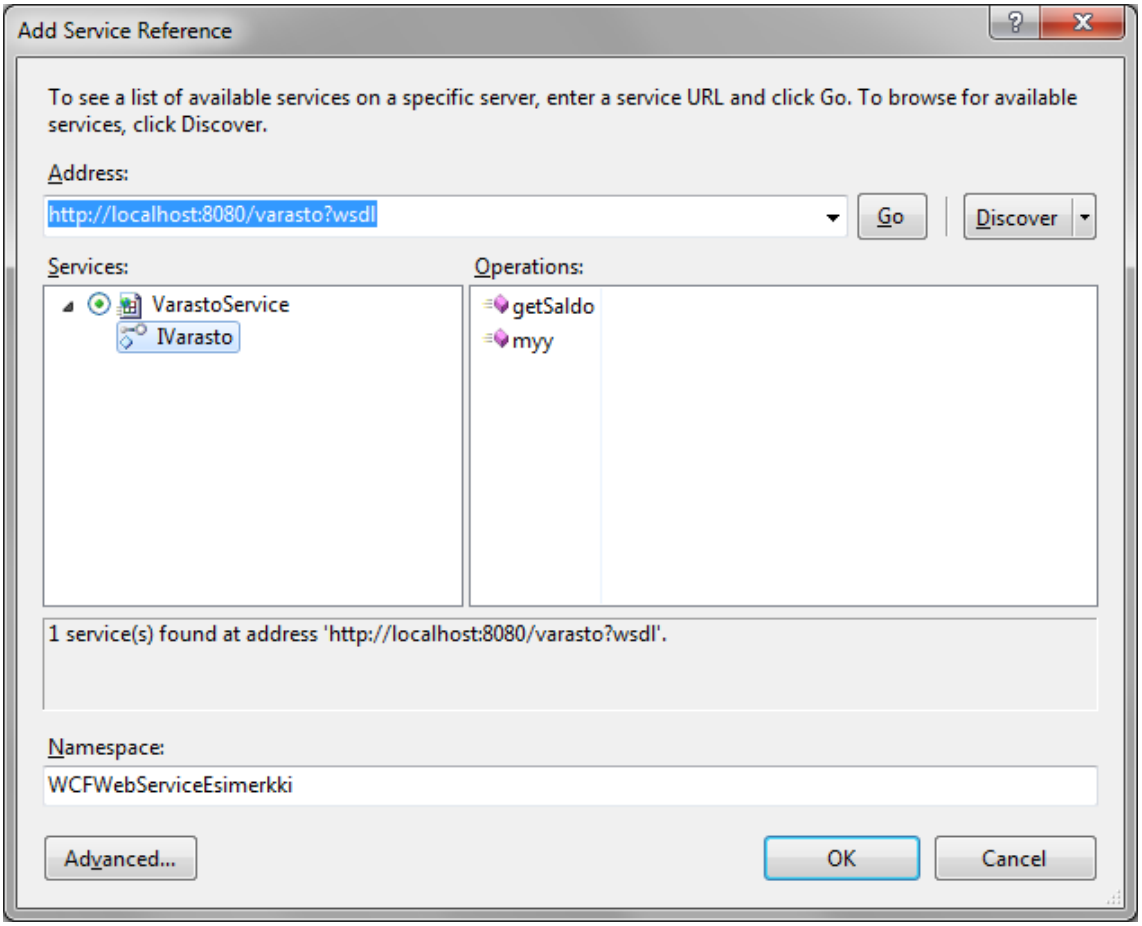
```
ServiceHost host = new ServiceHost(typeof(VarastoService), new
Uri("http://localhost:8080/varasto"));
            ServiceMetadataBehavior smb = new
ServiceMetadataBehavior() { HttpGetEnabled = true };
            host.Description.Behaviors.Add(smb);
host.Open();
```

Oheisessa esimerkissä on määritelty ServiceHost -tyyppinen objekti, joka ottaa parametreina tiedon julkaistavasta palvelusta sekä osoitteen minkä kautta palvelua voidaan käyttää. Lisäksi siinä on määritelty sisäistä toimintaa mahdollistamaan palvelun WSDL-dokumenttien haku.

Esimerkin lähdekoodi on kokonaisuudessaan liitteessä 2.

4.4.2 Palvelun käyttö asiakasohjelmalla

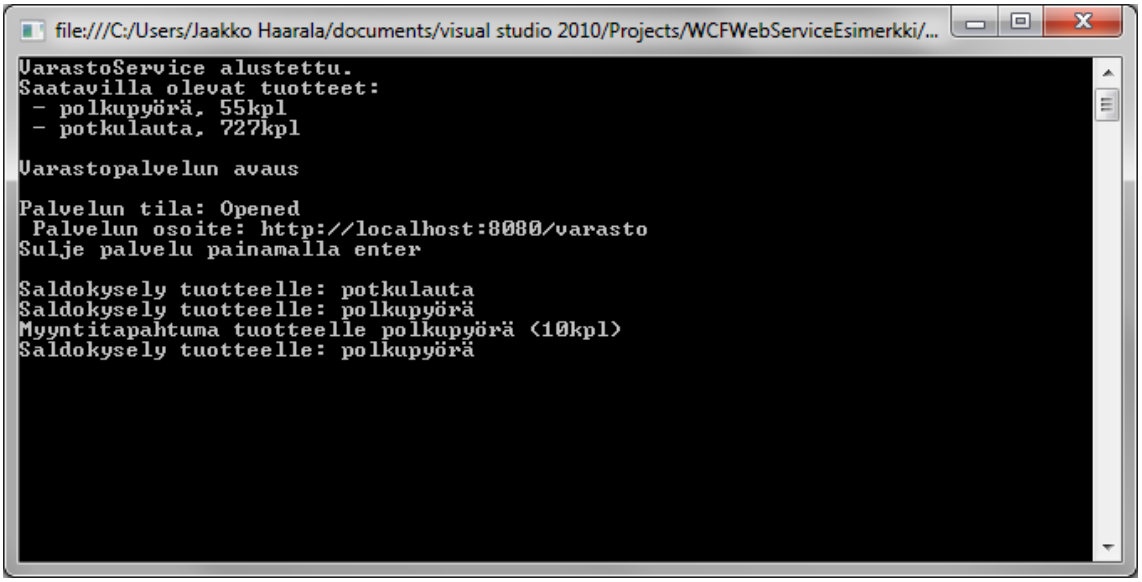
Jotta palvelua voidaan käyttää, tarvitsee sille luoda asiakasohjelmisto. Microsoft Visual Studio 2010:ssä tämä onnistuu käyttämällä Add Service Reference -toimintoa (kuva 1), joka generoi tarvittavat proxy-luokat annetun WSDL-dokumentin pohjalta.



Kuva 1. Visual Studion Add Service Reference -dialogi

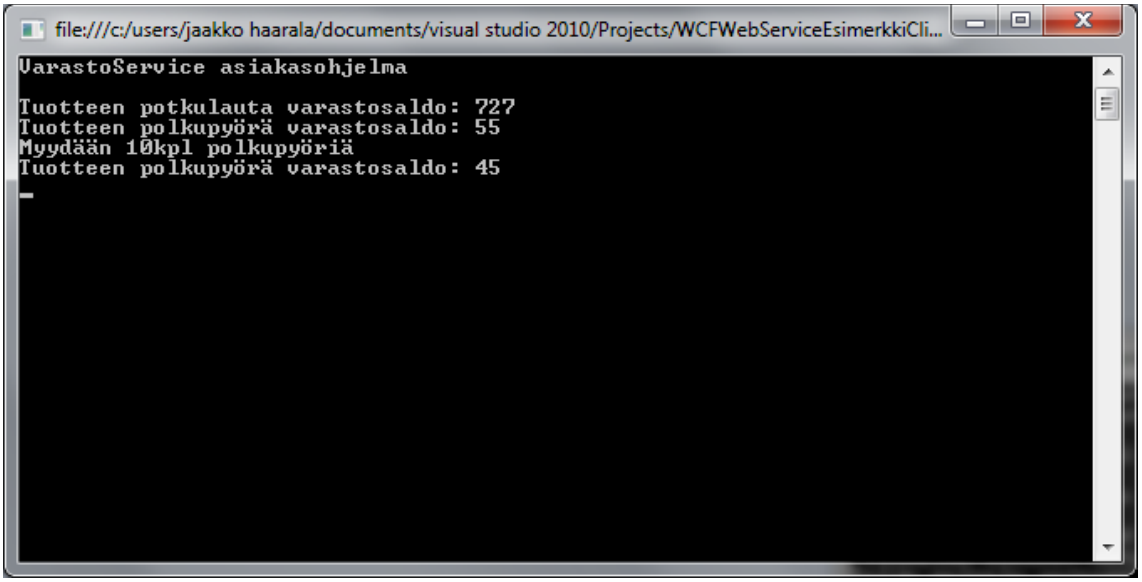
Kuvassa 1 ollaan generoimassa tarvittavia proxy-luokkia, joiden avulla voidaan kommunikoida aikaisemmin toteutetun varasto-palvelun kanssa. Asiakasohjelman lähdekoodi on kokonaisuudessaan liitteessä 3.

Asiakasohjelman toimintaa palvelun kanssa voidaan testata käynnistämällä ensin aikaisemmin toteutettu palvelu. Kun palvelu on toiminnassa, voidaan sille suorittaa kutsuja asiakasohjelmalla (kuva 2).



Kuva 2. Varastopalvelu

Kuvasta 2 käy ilmi varastossa olevista tuotteista ja siitä mitä kutsuja palvelu on vastaanottanut palvelua käyttäviltä asiakkailta (kuva 3).



Kuva 3. Varastopalvelun asiakas

4.4.3 Yhteenveto

Kuten esimerkistä käy ilmi, web service -pohjaisten palvelujen toteuttaminen koodikeskeisellä lähestymistavalla ei eroa huomattavasti tavanomaisesta ohjelmoinnista.

Eri toteutusteknologiat mahdollistavat erilaisia tapoja ja eri määrän mahdollisuuksia muokata palveluiden toimintaa. WCF-pohjaisissa tämä tapahtuu esimerkissäkin käytettyjen annotaatioiden avulla sekä XML-pohjaisilla konfiguraatiotiedostoilla.

5 KALENTERITIETOJEN INTEGROINTI

Tavoitteena oli suunnitella ja toteuttaa proof of concept -tyyppinen kalenteritietojen integrointi Sofokus iManager- ja Microsoft Exchange Server-järjestelmien välillä. Tarkoituksena oli saada tietoa siitä, miten integraatio on mahdollista toteuttaa ja minkälaisia tuotteistamismahdollisuuksia se mahdollistaa.

Vaatimusmäärittelyn (liite 4) tärkeimpinä vaatimuksina oli, että kalenteritietoja lisättäessä, poistaessa tai muokatessa muutokset näkyvät reaaliajassa molemmissa järjestelmissä, riippumatta siitä kumpaa järjestelmää käytetään.

5.1 Järjestelmien esittely

Sofokus iManager on Java-teknologioilla toteutettu, selainpohjainen projektinhallinta- ja toiminnanohjausjärjestelmä. Järjestelmässä ei ennestään ole web service rajapintoja kalenteritietojen käsittelyä varten, joka mahdollisti vapauden valita käyttötarkoitukseen sopivimman Java-pohjaisen teknologian integraation toteuttamiseksi.

Microsoft Exchange Server on monipuolinen viestintäratkaisu yrityksille, joka tarjoaa muun muassa sähköposti-, kalenteri-, yhteystieto- ja vastaajatoimintoja. Exchange Server sisältää versiosta 2007 lähtien web service rajapinnan, Exchange Web Services (EWS), joka mahdollistaa järjestelmän eri toimintojen käyttämisen ja tietojen käsittelyn (Microsoft Corporation 2006).

Microsoft Exchange Server sisältää lisäksi push- ja pull-notifikaatio -toiminnot, jotka mahdollistavan sen, että palvelimelta saadaan tarpeen mukaan reaaliajassa tietoa muutoksista määriteltyjen käyttäjien kalentereissa.

5.2 Lähestymistavat integraation toteutukseen

5.2.1 Vaihtoehdot

Järjestelmien välisen integraation toteuttamisvaihtoehtoja selvitettäessä tunnistettiin kolme erilaista tapaa toteuttaa integraatio web service teknologioilla:

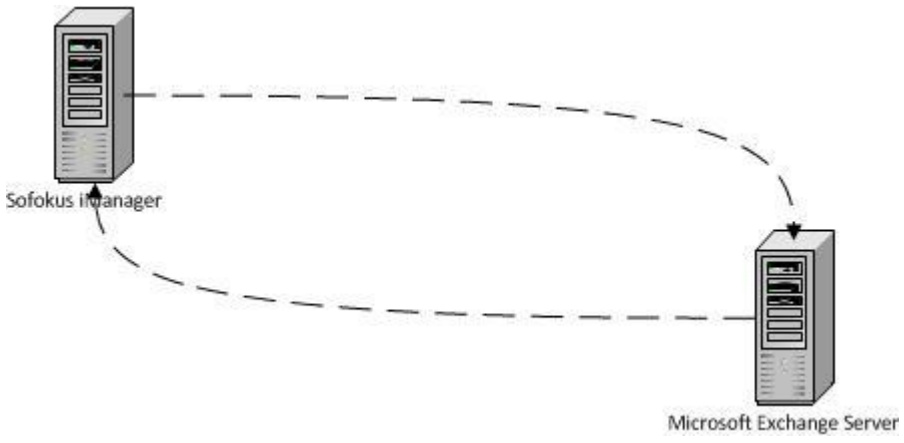
- Suora integraatio
- Epäsuora integraatio
- Erillinen Java EWS-kirjasto

Näistä kolmesta vaihtoehdosta tarkempaan tarkasteluun otettiin kaksi ensimmäistä vaihtoehtoa, joissa hyödynnetään Javan tai .NET frameworkin tarjoamia web service -toteutuksia.

Kolmas vaihtoehto, erillinen Java-pohjainen EWS-kirjasto, koettiin vaativan liian paljon resursseja ennen kuin saadaan tuloksia ja myöhemmässä ylläpidossa.

5.2.2 Suora integraatio

Ensimmäinen ja loogisimmalta kuulostava vaihtoehto oli toteuttaa suora integraatio järjestelmien välille. Suorassa integraatiossa (Kuvio 7) järjestelmät ovat suorassa viestiyhteydessä toisiinsa.



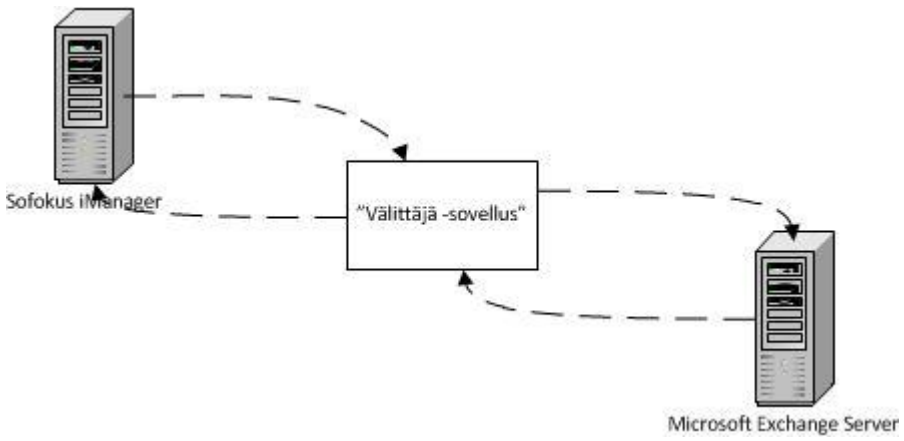
Kuvio 7. Suora integraatio

Suoran integraation etuja on sen yksinkertaisuus. Riittää, että järjestelmät saavat yhteyden toisiinsa ja niihin on toteutettu tarvittavat asiat yhteentoimivuuden takaamiseksi.

Ylläpidon kannalta suoran integraation etuja ovat helppo hallittavuus, koska tarvittavien konfiguraatioiden lisäksi ei tarvitse asentaa ylimääräisiä sovelluksia.

5.2.3 Epäsuora integraatio

Toinen vaihtoehto oli toteuttaa epäsuora integraatio (Kuvio 8), jossa järjestelmien välinen viestiliikenne kulkee tätä varten toteutetun integraatiosovelluksen kautta.



Kuvio 8. Epäsuora integraatio

Epäsuora integraatio on suoraan verrattuna hieman monimutkaisempi, koska siinä pitää ottaa huomioon molempien järjestelmien viestiliikenne välittäjänä toimivan sovelluksen kanssa.

Ylläpidon kannalta se aiheuttaa myös enemmän työtä asennuksen ja konfiguroinnin osalta. Tätä voidaan kuitenkin helpottaa esimerkiksi asennusohjelman avulla, jonka avulla asennuksen aikaa vieviä osuuksia voidaan helpottaa hoitamalla ne ohjelmallisesti.

Epäsuoran toteutuksen suurin etu on siinä, että se antaa vapauden tarvittavien rajapintojen määrittämiselle Sofokus iManagerin ja integraatiosovelluksen

välille. Lisäksi sovellus voi käsitellä Exchangelta tulevat tiedot sopivaan muotoon ennen niiden lähettämistä eteenpäin Sofokus iManagerille.

.NET Framework-pohjaisen integraatiosovelluksen toteutus antaa myös mahdollisuuden hyödyntää Exchange Web Services Managed API:a (Myöhemmin EWS Managed API). EWS Managed API on Microsoftin tekemä ja ylläpitämä kirjasto .NET -pohjaisille sovelluksille, joka yksinkertaistaa EWS:n käyttöä. (Microsoft Corporation 2010b.)

5.3 Suoran integraation prototyyppi

5.3.1 Alkutilanne ja tavoite

Toteutusvaihtoehtojen pohjalta lähdettiin aluksi toteuttamaan Java-pohjaista prototyyppiä suoralle integraatiolle. Prototyypin tarkoituksena oli hakea tietyn käyttäjän kalenteritietoja Exchange -palvelimelta.

5.3.2 Toteutustavan ongelmat

Prototyypin toteutuksessa törmättiin heti alussa ongelmiin yritettäessä generoida tarvittavat proxy-luokat EWS:ää varten Java Metron mukana tulevalla työkalulla.

Proxy-luokkien generointi työkalulla epäonnistuu, koska Exchange Serverin julkaisemasta WSDL-dokumentista puuttuu kokonaan endpoint -määrittely, joka sisältää tiedon palveluun voidaan olla yhteydessä (W3C 2001).

Ongelma pystyttiin kiertämään lataamalla Exchange Serverin julkaisema WSDL-dokumentti ja lisäämällä dokumenttiin puuttuva määrittely, jonka jälkeen proxy-luokkien generoiminen on mahdollista.

Muokatun WSDL-dokumentin avulla generoidut proxy-luokat mahdollistivat endpointiksi määritellyn Exchange-palvelimen kanssa kommunikoinnin ja käyttäjän kalenteritietojen käsittelyn.

Toteutuksessa huomattiin kuitenkin kaksi muuta ongelmaa, jotka liittyivät Java Metron tapaan toimia.

EWS hyväksyy null -arvojen antamisen web service -kutsujen parametreina, mutta jax-ws ei hyväksy näiden lähettämistä. Tästä syystä tarvittavista metodeista joudutaan tekemään ylikuormitetut versiot, joiden parametreista on jätetty pois ne arvot jotka halutaan jättää kutsussa null:ksi.

Toinen havaittu ongelma liittyy siihen, kun halutaan määrittää ohjelmallisesti uusi endpoint, esimerkiksi toinen Exchange-palvelin. Endpointin määrittäminen ohjelmallisesti aiheuttaa sen, että Java Metro yrittää tarkistaa palvelimen wsdl-dokumentin, joka johtaa lopulta virheeseen koska palvelimen wsdl-dokumentista puuttuu aiemmin mainittu endpoint -määrittely.

Eli vaikka tarvittavat luokat pystytään generoimaan käsin muokatusta WSDL-dokumentista, aiheuttaa se ongelmia muun muassa tuotteistamisen kannalta, koska Java Metron toteutuksen takia muokkaukset pitäisi tehdä joka kerta erikseen jokaiselle Exchange-palvelimelle, jonka kanssa halutaan kommunikoida.

5.3.3 Päätelmät jatkoa varten

Vaikka Java-pohjainen prototyyppi saatiin toimimaan Exchangen kanssa, toteutuksessa vastaan tulleiden ongelmien takia päädyttiin hylkäämään ajatus suorasta integraatiosta, ja tämän sijaan toteuttamaan integraatio epäsuorasti erillisellä .NET framework -pohjaisella integraatiosovelluksella.

5.4 Epäsuoran integraation toteutus

5.4.1 Suunnittelu ja toteutuksen eteneminen

Integraation toteuttamiseen integraatiosovelluksen avulla käytännössä oli muutamia vaihtoehtoja. Suunnittelun lähtökohdiksi otettiin se, että sovellus olisi toiminnaltaan mahdollisimman yksinkertainen ja toimintavarma. Sen tulisi siis toimia vain välittäjänä, joka käsittelee kalenteritiedot järjestelmien

ymmärtämään muotoon. Esimerkiksi toimintansa kannalta oleelliset tiedot, kuten esimerkiksi minkä käyttäjien tietoja tulisi seurata, sovellus hakisi suoraan Sofokus iManagerilta, eikä niitä tarvitse antaa sille erikseen.

Integraatiosovelluksen toteutus eteni vaiheittain. Alun suunnittelun ja toteutusteknologioiden valitsemisen jälkeen rajapintojen operaatiot ja sovelluksen ominaisuudet toteutettiin yksi kerrallaan. Kun ominaisuus saatiin valmiiksi, sen toiminta testattiin ja varmistettiin oikeaksi ennen seuraavan ominaisuuden toteuttamiseen siirtymistä. Lisäksi toteutuksen edetessä tarkistettiin tarpeen vaatiessa toteutussuunnitelmaa, hyvän lopputuloksen varmistamiseksi.

5.4.2 Käytetyt teknologiat

Integraatiosovelluksen toteutuksessa käytettiin vuoden 2010 alkupuolella julkaistua .NET frameworkin versiota 4. Sen uudistuksiin kuuluu muun muassa säieturvalliset collection -luokat. Kyseisiä luokkia hyödyntämällä sovelluksen toteutus pystyttiin pitämään yksinkertaisempana, koska sovelluksen toiminnan kannalta monisäikeistyksestä huomioon otettavat asiat ovat näissä valmiiksi toteutettuna.

Web service toimintojen osalta sovelluksessa hyödynnettiin Windows Communication Foundationia, joka on ollut osa .NET frameworkia sen versiosta 3.0 lähtien.

Sofokus iManagerin osalta käytettiin hyväksi Java Metro web service stackia. Metro on hyvä vaihtoehto kun halutaan kommunikoida WCF-pohjaisten palvelujen kanssa, koska sen yksi päätavoitteista on mahdollisimman hyvä yhteentoimivuus WCF-teknologioiden kanssa (GlassFish Project 2010).

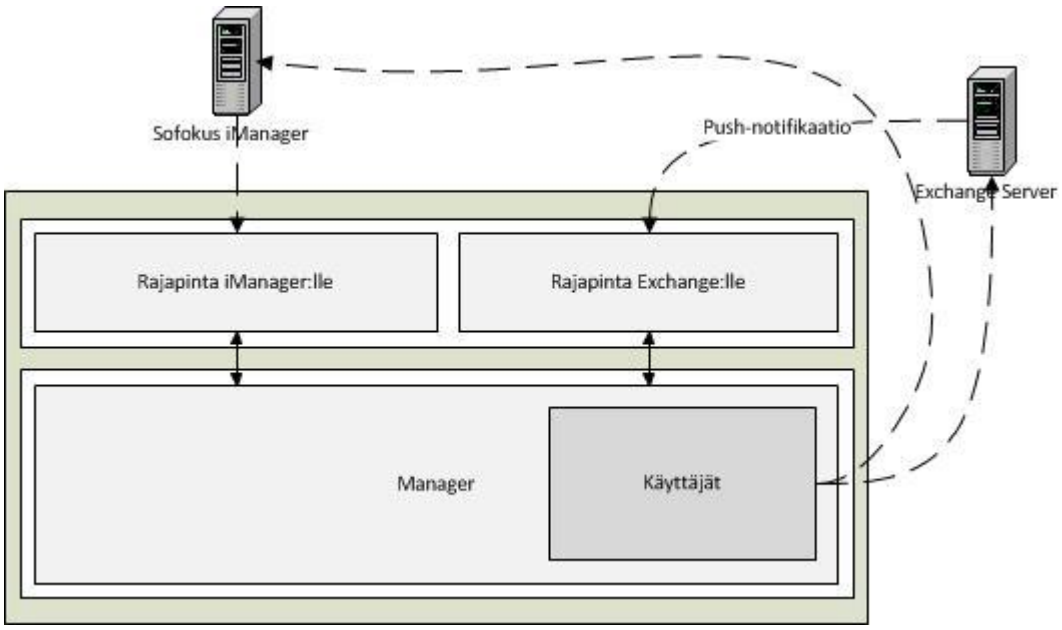
5.4.3 Sovelluksen toiminta

Exchange Serverin ja integraatiosovelluksen välisessä kommunikaatiossa hyödynnetään Exchange Serverin tarjoamia web service rajapintoja.

Sovelluksen toiminta perustuu pitkälti niin sanottujen push-notifikaatioiden hyödyntämiseen.

5.4.4 Sisäinen rakenne

Sovelluksen sisäinen rakenne koostuu web service -rajapinnoista, käyttäjätiedoista ja managerista joka huolehtii henkilöiden kalenteritietojen päätymisestä oikeisiin kohteisiin. Sovelluksen rakennetta ja kommunikaatiota järjestelmien välillä on havainnollistettu yleisellä tasolla kuviossa 9.



Kuvio 9. Sovelluksen rakenne

Sovelluksen toiminta on jaettu karkeasti kahteen osaan. Web service -rajapinta on sovelluksen ulospäin näkyvä osa, jonka kautta vastaanotetut pyynnöt välitetään sovelluksen sisäisille osille.

Vastaanotetut kutsut käsitellään ja kohdennetaan managerin ylläpitämien käyttäjätietojen avulla oikeille käyttäjätileille molemmissa järjestelmissä.

5.4.5 Suorituskyky

Integraatiosovelluksen toteutuksessa pyrittiin huomioimaan myös suorituskyky. Sovelluksen tulisi voida vastaanottaa ja käsitellä viestejä edelleen lähetettäväksi mahdollisimman pienellä viiveellä.

Viivettä on pyritty integraatiosovelluksessa vähentämään käyttämällä luvussa 4.4 esitellyn esimerkkipalvelun tavoin yhtä instanssia palvelun implementoivasta luokasta jolloin vältytään aikaa vieviltä alustustoimenpiteiltä (initialisointi).

Oletuksena yksi instanssi voi suorittaa kerralla vain yhden toimenpiteen alusta loppuun. Integraatiosovelluksen tapauksessa tämä toiminta ei kuitenkaan ole suotavaa, koska se aiheuttaa viiveitä kun palvelukutsuja tulee useampia samaan aikaan. Käytännössä hyvän suorituskyvyn takaamiseksi sovelluksen tulisi siis pystyä käsittelemään vastaanotettuja web service kutsuja rinnakkain sitä mukaan, kuin niitä vastaanotetaan.

WCF:n annotaatioita käyttämällä on mahdollista muuttaa palvelun toimintaa niin, että yksi instanssi voi suorittaa useita toimia rinnakkain. Tällöin pitää huolehtia enää rinnakkaisuudesta aiheuttamista ongelmista.

Integraatiosovelluksessa pyyntöjen rinnakkain suorittamisen ongelmat liittyivät lähinnä tietojen lukemiseen ja tallentamiseen sovelluksen sisällä. Ongelma ratkaistiin käyttämällä säieturvallisia luokkia niiltä osin kuin tarpeellista. Koska Exchange Server lähettää tietyn käyttäjien muutokset sitä mukaan kun niitä saadaan prosessoitua, toteutuksessa ei tarvinnut erikseen huolehtia siitä, ettei vanhempi tieto korvaa uudempaa tietoa.

5.5 Web service rajapintojen määrittely

5.5.1 Lähestymistapa

Web service rajapintoja lähdettiin määrittelemään koodin kautta, eli lähestymistapa oli koodikeskeinen. Koodikeskeiseen lähestymistapaan

päädyttiin sen helppouden takia. Lisäksi katsottiin WCF:n ja Java Metron tarjoavan tarpeeksi joustavuutta, mikäli halutaan tulevaisuudessa muokata sisäistä toteutusta ja samalla varmistaa, että web service -rajapinta ei muutu.

Microsoft Exchange Server SDK:n mukana toimitetaan muun muassa WSDL-dokumentin, joka määrittelee tarvittavat rajapinnat push-notifikaatioiden vastaanottamista varten Exchange Serveriltä. Tältä osin hyödynnettiin dokumenttikeskeistä lähestymistapaa ja rajapinnat generoitiin WSDL-dokumentin pohjalta.

5.5.2 Integraatiosovelluksen rajapinnat

Integraatiosovelluksen rajapinnat koostuvat itse määritellyistä metodeista, jotka ovat tarkoitettu kommunikoinnin vastaanottamiseen iManagerilta, ja push-notifikaatioiden vastaanottamiseen tarkoitetusta rajapinnasta.

Vaikka rajapinnat koostuvat kahdesta eri määrittelystä, julkaistaan ne silti yhtenä web-palveluna, koska tämä pitää sovelluksen rakennuksen yksinkertaisempana ja vähentää toistoa.

Liitteessä 5 on ote rajapinnan määrittelystä, jota tarkastelemalla voidaan todeta, että rajapintojen yhdistäminen yhdeksi palveluksi on toteutettu periyttämällä lopulta käytettävä rajapinta Exchange Serverin push-notifikaatiota varten luodusta proxy-luokasta.

5.5.3 Sofokus iManagerin rajapinta

iManageria varten määriteltiin niin ikään Java Metroon perustuva rajapinta. Rajapinta sisältää tarvittavat toiminnot käyttäjien kalenterien kanssa toimimiseen.

Tämän lisäksi se sisältää toimintoja, joita kutsumalla integraatiokomponentti voi pyytää sen toiminnan kannalta oleellisia tietoja. Tällaisia tietoja ovat esimerkiksi käyttäjät joiden kalenteritietojen muutoksia tulisi seurata.

Liitteessä 6 on ote rajapinnan määrittelystä. Liitteestä käy ilmi kuinka WCF:n tavoin Metro-pohjaisissa palveluissa palvelun toimintaa voidaan muokata annotaatioiden avulla.

5.5.4 Parhaiden käytäntöjen hyödyntäminen

Rajapintojen määrittelyssä pyrittiin ottamaan huomioon rajapintojen versiointiin liittyviä seikkoja, jotka ovat tärkeitä silloin, kun rajapintoihin joudutaan tekemään niin sanottuja rikkovia muutoksia (breaking change). Versiointi toteutettiin käyttämällä namespace -määrittelyissä päivämääriä, joista käy ilmi milloin kyseiseen rajapinnan osaan on tehty muutoksia (Microsoft Corporation, 2010a).

5.5.5 Tietoturva

Tietoturvan osalta toteutuksessa pyrittiin pitämään tiedonsiirto turvallisena, mutta samalla mahdollisimman yksinkertaisena mahdollisilta ongelmilta välttymiseksi.

Integraation tietoturvan takaamiseksi päädyttiin siirtopohjaiseen toteutukseen, jossa tiedonsiirto tapahtuu salatun HTTPS-protokollan kanssa. Tämän lisäksi toteutettiin tunnistusmekanismi hyödyntämällä SOAP-viestien header-osiota, jonka avulla Sofokus iManager ja integraatiosovellus pystyvät tunnistamaan toisensa.

6 YHTEENVETO

Web service -pohjaiset palvelut ovat hyvä vaihtoehto kun halutaan tarjota rajapintaa, jonka kautta on mahdollista käyttää järjestelmän toimintoja tai tietoja verkon yli.

Java Metrolla ja Windows Communication Foundationilla toteutetut web servicet toimivat opinnäytteen työstämisen aikana tehtyjen havaintojen perusteella hyvin yhteen, kun toteutukset pidetään tarpeeksi yksinkertaisina.

Tämä oli osin odotettua, koska Java Metron kehityksessä on pyritty yhteentoimivuuteen WCF:n kanssa. Eroavaisuuksia toiminnassa alkaa tulla vastaan siinä vaiheessa, kun pitää monimutkaistaa palveluita, esimerkiksi tietoturvan osalta ottamalla käyttöön viestipohjaisessa tietoturvassa käytettäviä tekniikoita.

Opinnäytteen toiminnallisessa osuudessa toteutettu integraatiosovellus on hyvä esimerkki siitä, miten on mahdollista toteuttaa kahden, eri alustoihin perustuvan järjestelmän kalenteritietojen integrointi.

Se antaa myös kuvaa, minkälaisia haasteita voi tulla vastaan isojen ja monimutkaisten järjestelmien, kuten Microsoft Exchange Server, web service rajapintoja käytettäessä, joissa on jouduttu tekemään tavallisuudesta poikkeavia ratkaisuja. Niin ikään se korostaa eroja eri web service -toteutusten välillä. Kuten luvusta 5.4 käy ilmi, Java Metro on hyvin tarkka siitä, että palvelumäärittelyssä on kaikki vaaditut elementit.

Toiminnaltaan toteutettu proof of concept -integraatiosovellus vastasi hyvin vaatimusmäärittelyssä asetettuja tavoitteita. Tuotantokäyttöä ajatellen se vaatii kuitenkin vielä jatkokehitystä rajapintojen osalta, jotka eivät mahdollista vielä kovin yksityiskohtaisia parametreja siitä, mitä halutaan.

Web service -pohjaisten palvelujen toteuttamiseen on olemassa koodikeskeisen lähestymistavan osalta hyvin erilaisia työkaluja. Microsoftin Visual Studio -kehitysympäristö tarjoaa hyvän ja helpon tavan .NET framework -pohjaisten palvelujen toteuttamiseen.

Java-alustalla palveluja voidaan toteuttaa esimerkiksi Eclipse IDE:llä ja sopivilla laajennoksilla sekä Net Beans IDE:llä. Useiden kehitysympäristöjen lisäksi on mahdollisuus valita useista eri web service -toteutuksista, kuten Metro jota tässä opinnäytteessä on käytetty. Metron lisäksi muita toteutuksia ovat esimerkiksi Apache Axis ja Apache CXF.

Opinnäytteessä pysyttiin web service -teknologioiden yhteentoimivuuden osalta hyvin yleisellä tasolla aiheen laajuudesta johtuen, eikä se ottanut kantaa eri web service -implementaatioiden keskinäisiä eroja esimerkiksi standarditukien osalta. Jatkotutkimuksen kannalta aihe on otollinen, koska implementaatioita on olemassa useita ja koska web servicet ovat oleellinen osa nykyaikaisten järjestelmien välistä tietoliikennettä.

LÄHTEET

Colan, M. 2004. Service-Oriented Architecture expands the vision of Web services, Part 1. Viitattu 17.7.2010 http://www.ibm.com/developerworks/webservices/library/ws-soaintro.html

Dhesiaseelan, A. 2004. What's New in WSDL 2.0. Viitattu 12.11.2010 http://www.xml.com/pub/a/ws/2004/05/19/wsdl2.html

Finanssialan Keskusliitto. 2010. Palvelukuvaukset: Tietoturva ja asiakasyhteydet. Viitattu 18.7.2010 http://www.fkl.fi/www/page/fk_www_3830

GlassFish Project. 2010. How Metro Relates to .NET Windows Communication Foundation (WCF). Viitattu 7.11.2010 https://metro.dev.java.net/guide/How_Metro_Relates_to__NET_Windows_Communication_Foundation__WCF_.html

IBM Corporation. 2004. Web Services Security. Viitattu 14.11.2010 http://www.ibm.com/developerworks/library/specification/ws-secure/

IEEE. The Institute of Electrical and Electronic Engineers. 1991. IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries. New York : The Institute of Electrical and Electronic Engineers.

IETF. The Internet Engineering Task Force. 1996. Hypertext Transfer Protocol -- HTTP/1.0. Viitattu 15.11.2010 http://tools.ietf.org/html/rfc1945#page-48

IETF. The Internet Engineering Task Force. 1997. An Extension to HTTP : Digest Access Authentication. Viitattu 15.11.2010 http://tools.ietf.org/html/rfc2069

ISO/IEC 2382-1. Information Technology Vocabulary, Fundamental Terms.

Kalin, M. 2009. Java Web Services: Up and Running. Sebastopol : O'Reilly Media.

Microsoft Corporation. 2002. Understanding WS-Security. Viitattu 28.11. http://msdn.microsoft.com/en-us/library/ms977327.aspx

Microsoft Corporation. 2006. Microsoft TechNet. Viitattu 17.7. http://technet.microsoft.com/en-us/library/aa998811(EXCHG.80).aspx

Microsoft Corporation. 2010a. Best Practices. Data Contract Versioning. Viitattu 28.11.2010 http://msdn.microsoft.com/en-us/library/ms733832.aspx

Microsoft Corporation. 2010b. Introducing the Exchange Web Services Managed API 1.0. Viitattu 7.11.2010 http://msdn.microsoft.com/en-us/library/dd637749(EXCHG.80).aspx

Microsoft Corporation. 2010c. Metro to WCF Interoperability. Viitattu 21.11.2010 http://msdn.microsoft.com/en-us/library/ff842400.aspx

Microsoft Corporation. 2010d. Web Services Protocols Interoperability Guide. Viitattu 14.11.2010 http://msdn.microsoft.com/en-us/library/ms734776.aspx

Resnick, S.; Crane, R. & Bowen, C. 2008. Essential Windows Communication Foundation: For .NET Framework 3.5. Upper Saddle River : Addison-Wesley.

Sanastokeskus TSK ry. 2005. TEPA-termipankki. Viitattu 30.10.2010 http://www.tsk.fi/tepa/

Sanastokeskus TSK ry. 2009. Tietotekniikan termitalkoot. Viitattu 31.10.2010 http://www.tsk.fi/tsk/termitalkoot/hakemistot-267.html?page=get_id&id=ID0176&vocabulary_code=TSKTT

Twitter Inc. 2010. Twitter API Wiki. Viitattu 17.17.2010 http://apiwiki.twitter.com/API-Overview

W3C. World Wide Web Consortium. 2001. Web Services Description Language (WSDL). Viitattu 7.11.2010 http://www.w3.org/TR/wsdl

W3C. World Wide Web Consortium. 2004. Web Services Architecture. Viitattu 2.9.2010 http://www.w3.org/TR/ws-arch/

W3C. World Wide Web Consortium. 2007. SOAP Version 1.2 Part 1: Messaging Framework. Viitattu 3.10.2010 http://www.w3.org/TR/soap12-part1/

W3C. World Wide Web Consortium. 2010. About W3C. Viitattu 2.10.2010 http://www.w3.org/Consortium/

Wikipedia. 2010. Web Services Description Language. Viitattu 14.11.2010 http://en.wikipedia.org/wiki/Web_Services_Description_Language

WS-I. Web Services Interoperability Organization. 2009. About WS-I. Viitattu 21.11.2010 http://www.ws-i.org/about/Default.aspx

Esimerkki WSDL-dokumentista

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="VarastoService"
   targetNamespace="http://palvelu/wsdl/VarastoService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://palvelu/wsdl/VarastoService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SaldoRequest">
      <part name="tuote" type="xsd:string"/>
   </message>
   <message name="SaldoResponse">
      <part name="saldo" type="xsd:string"/>
   </message>

   <portType name="Varasto_PortType">
      <operation name="Saldo">
         <input message="tns:SaldoRequest"/>
         <output message="tns:SaldoResponse"/>
      </operation>
   </portType>

   <binding name="Varasto_Binding" type="tns:Varasto_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="Saldo">
         <soap:operation soapAction="Saldo"/>
         <input>
            <soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:VarastoService"
               use="encoded"/>
         </input>
         <output>
            <soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:VarastoService"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Varasto_Service">
      <documentation>Varastopalvelun WSDL-dokumentti</documentation>
      <port binding="tns:Varasto_Binding" name="Varasto_Port">
         <soap:address
            location="http://localhost:8080/varasto"/>
```

```
      </port>
   </service>
</definitions>
```

WCF web service esimerkki

```
namespace WCFWebServiceEsimerkki
{
    /// <summary>
    /// Rajapintojen määrittely
    /// </summary>
    [ServiceContract(Namespace="http://VarastoEsimerkki/")]
    interface IVarasto
    {
        [OperationContract]
        int getSaldo(string tuote);

        [OperationContract]
        void myy(string tuote, int maara);
    }

    /// <summary>
    /// Käytännön toteutus määritellylle rajapinnalle.
    /// </summary>
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, Name =
"VarastoService", Namespace = "http://VarastoEsimerkki/")]
    class VarastoService : IVarasto
    {
        private Dictionary<string, int> tuotteet = new Dictionary<string, int>();

        VarastoService()
        {
            Random rand = new Random();
            tuotteet.Add("polkupyörä", rand.Next(0,1000));
            tuotteet.Add("potkulauta", rand.Next(0, 1000));
            Console.WriteLine("VarastoService alustettu. \nSaatavilla olevat
tuotteet:");
            foreach (var tuote in tuotteet)
            {
                Console.WriteLine(" - {0}, {1}kpl", tuote.Key, tuote.Value);
            }
        }

        public int getSaldo(string tuote)
        {
            Console.WriteLine("Saldokysely tuotteelle: {0}", tuote);
            if (tuotteet.ContainsKey(tuote))
            {
                return tuotteet[tuote];
            }
            else
            {
                throw new KeyNotFoundException("Väärä tuotenimi.");
            }
        }

        public void myy(string tuote, int maara)
        {
            Console.WriteLine("Myyntitapahtuma tuotteelle {0} ({1}kpl)", tuote,
maara);
            if (tuotteet.ContainsKey(tuote))
            {
                tuotteet[tuote] = tuotteet[tuote] - maara;
```

```
            }
            else
            {
                throw new KeyNotFoundException("Väärä tuotenimi.");
            }
        }
    }

    /// <summary>
    /// Pääohjelma
    /// </summary>
    class VarastoServiceHost
    {
        static void Main(string[] args)
        {
            // Palvelun tiedot ja asetukset
            ServiceHost host = new ServiceHost(typeof(VarastoService), new
Uri("http://localhost:8080/varasto"));
            ServiceMetadataBehavior smb = new ServiceMetadataBehavior() {
HttpGetEnabled = true }; // Mahdollistaa mm. palvelun wsdl-dokumentin tutkimisen
            host.Description.Behaviors.Add(smb);

            // Palvelun julkaiseminen
            try
            {
                Console.WriteLine("\nVarastopalvelun avaus \n");
                host.Open();
                Console.WriteLine("Palvelun tila: {0} \n Palvelun osoite: {1}",
host.State, host.BaseAddresses.First().AbsoluteUri);
            }
            catch (Exception e)
            {
                Console.WriteLine("Virhe palvelun julkaisemisessa: {0}",
e.Message);
            }

            Console.WriteLine("Sulje palvelu painamalla enter \n");
            Console.ReadLine();

            // Palvelun sulku
            host.Close();
            Console.WriteLine("Palvelun tila: {0}", host.State);

        }
    }

}
```

WCF web service client esimerkki

```
namespace WCFWebServiceEsimerkkiClient
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("VarastoService asiakasohjelma\n");
            VarastoClient serviceClient = new VarastoClient();
            serviceClient.Open();

            Console.WriteLine("Tuotteen     potkulauta     varastosaldo:     {0}",
             serviceClient.getSaldo("potkulauta"));

            // polkupyörien määrä ennen ja jälkeen myyntitapahtuman
            Console.WriteLine("Tuotteen     polkupyörä     varastosaldo:     {0}",
             serviceClient.getSaldo("polkupyörä"));
            Console.WriteLine("Myydään 10kpl polkupyöriä");
            serviceClient.myy("polkupyörä", 10);
            Console.WriteLine("Tuotteen     polkupyörä     varastosaldo:     {0}",
             serviceClient.getSaldo("polkupyörä"));

            Console.ReadLine();
            serviceClient.Close();
        }
    }
}
```

Kalenteri-integraation vaatimusmäärittely

Tavoite

Tavoite on toteuttaa proof of concept -sovellus, joka mahdollistaa kalenteritietojen integroinnin Sofokus iManager- ja Microsoft Exchange Server -järjestelmien välillä.

Vaatimukset

Sovelluksen tulee toiminnaltaan täyttää seuraavia toiminnallisia ja ei toiminnallisia vaatimuksia.

Sovelluksen tulee pystyä synkronoimaan seuraavat toimenpiteet järjestelmien välillä:

- Kalenteritietojen käsittely
  - lisäys
  - poisto
  - päivitys
  - siirto
- Kalenterien käsittely
  - lisäys
  - poisto
  - päivitys

Muita vaatimuksia sovellukselle ovat:

- Tietojen reaaliaikainen päivittyminen kumpaankin järjestelmään
- Pystyy palvelemaan useampia käyttäjiä samaan aikaan
- Tulee toimia Microsoft Exchange Serverin versioilla 2007 ja 2010
- Tiedonsiirron pitää tapahtua turvallisesti

Esimerkkikäyttötapaukset

Puhelimella lisätyn kalenterimerkinnän synkronointi

1. Alkutilanne: Käyttäjä lisää matkapuhelimella merkinnän Exchangessa sijaitsevaan kalenteriin.

2. Tapahtuma tunnistetaan
3. Merkintä lisätään käyttäjän vastaavaan kalenteriin iManager:ssa

Toisen käyttäjän kalenterin päivitys

1. Käyttäjä lisää merkinnän toisen käyttäjän kalenteriin
    a. iManager:n kautta
    b. Exchange:n kautta
2. Tapahtuma tunnistetaan
3. Merkinnän tiedot lisätään toiseen järjestelmään

Julkisen kalenterin päivitys

Julkinen kalenteri voi olla esimerkiksi neuvotteluhuoneen varaukset sisältävä kalenteri.

1. Käyttäjä lisää merkinnän julkiseen kalenteriin
    a. iManager:n kautta
    b. Exchange:n kautta
2. Tapahtuma tunnistetaan
3. Merkintä lisätään toiseen järjestelmään vastaavaan kalenteriin

Ote integraatiosovelluksen web service rajapinnan määrittelystä

```
[ServiceContract(Namespace =
"http://sofokus.com/schemas/imgr/exch/2010/04/08")]
    interface IConnectorService : NotificationServicePortType
    {
        [OperationContract(Name = "addSubscription")]
        void addSubscription(User user);

        [OperationContract(Name="removeSubscription")]
        void removeSubscription(string userId);

        [OperationContract(Name="clearSubscriptions")]
        void clearSubscriptions();

        [OperationContract(Name = "syncFolders")]
        CalendarFolder[] syncronizeFolders(User user);

        [OperationContract(Name = "syncItems")]
        CalendarItem[] syncronizeItems(User user);

        [OperationContract(Name = "getTrashFolder")]
        CalendarFolder getTrashFolder(User user);

        [OperationContract(Name = "addCalendarItem")]
        void addCalendarItem(CalendarItem item);

        [OperationContract(Name = "removeCalendarItem")]
        void removeCalendarItem(string itemId);

        [OperationContract(Name = "updateCalendarItem")]
        void updateCalendarItem(CalendarItem item);

        [OperationContract(Name = "addCalendarFolder")]
        void addCalendarFolder(CalendarFolder folder);

        [OperationContract(Name = "removeCalendarFolder")]
        void removeCalendarFolder(string folderId);

        [OperationContract(Name = "updateCalendarFolder")]
        void updateCalendarFolder(CalendarFolder folder);
    }
```

Ote Sofokus iManagerin web service rajapinnan määrittelystä

```
@WebService(name = "CalendarService", targetNamespace =
"http://sofokus.com/schemas/imgr/exch/2010/04/21")
public interface Calendar {

           /**
            * Kalenterin poisto.
            * @param itemId Kalenterin id exchangessa.
            */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/r
emoveCalendarItem")
    public void removeCalendarItem(
        @WebParam(name = "itemId")
        String itemId);

    /**
     * Kalenterimerkinnän lisäys.
     * @param item Merkintä.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/a
ddCalendarItem")
    public void addCalendarItem(
        @WebParam(name = "item")
        CalendarItem item);

    /**
     * Merkinnän siirto kalenterista toiseen.
     * Huom: Siirron yhteydessä merkinnän Exchange id vaihtuu.
     * @param itemId Merkinnän nykyinen Exchange id.
     * @param targetFolderId Kalenterin Exchange id johon merkintä
siirretään.
     * @param newItemId Merkinnän uusi Exchange id.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/m
oveCalendarItem")
    public void moveCalendarItem(
        @WebParam(name = "itemId")
        String itemId,
        @WebParam(name = "targetFolderId")
        String targetFolderId,
        @WebParam(name = "newItemId")
        String newItemId);

    /**
     * Kalenterimerkinnän kopiointi
     * @param itemId Kopioitavan merkinnän Exchange id.
```

```
     * @param targetFolderId Kalenterin Exchange id, johon kopio
sijoitetaan.
     * @param copyItemId Kopion Exchange id.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/c
opyCalendarItem")
    public void copyCalendarItem(
        @WebParam(name = "itemId")
        String itemId,
        @WebParam(name = "targetFolderId")
        String targetFolderId,
        @WebParam(name = "copyItemId")
        String copyItemId);

    /**
     * Kalenterimerkinnän päivitys.
     * @param item Merkinnän tiedot.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/u
pdateCalendarItem")
    public void updateCalendarItem(
        @WebParam(name = "item")
        CalendarItem item);

    /**
     * Kalenterin lisäys.
     * @param folder Kalenterin tiedot.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/a
ddCalendarFolder")
    public void addCalendarFolder(
        @WebParam(name = "folder")
        CalendarFolder folder);

    /**
     * Kalenterin poisto.
     * @param folderId Kalenterin Exchange id.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/r
emoveCalendarFolder")
    public void removeCalendarFolder(
        @WebParam(name = "folderId")
        String folderId);

    /**
     * Kalenterin siirto.
     * @param folderId Siirrettävän kalenterin Exchange id.
```

```
     * @param targetFolderId Kohdekalenteri johon kalenteri
siirretään. (ei käytössä)
     * @param newFolderId Kalenterin uusi Exchange id.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/m
oveCalendarFolder")
    public void moveCalendarFolder(
        @WebParam(name = "folderId")
        String folderId,
        @WebParam(name = "targetFolderId")
        String targetFolderId,
        @WebParam(name = "newFolderId")
        String newFolderId);

    /**
     * Kalenterin kopiointi.
     * @param folderId Kopioitavan kalenterin Exchange id.
     * @param targetFolderId Kohdekalenteri johon kopio sijoitetaan.
(ei käytössä)
     * @param copyFolderId Kopion Exchange id.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/c
opyCalendarFolder")
    public void copyCalendarFolder(
        @WebParam(name = "folderId")
        String folderId,
        @WebParam(name = "targetFolderId")
        String targetFolderId,
        @WebParam(name = "copyFolderId")
        String copyFolderId);

    /**
     * Kalenterin tietojen päivitys.
     * @param folder Kalenterin tiedot.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/u
pdateCalendarFolder")
    public void updateCalendarFolder(
        @WebParam(name = "folder")
        CalendarFolder folder);

    /**
     * Palauttaa listan käyttäjistä, joille on tilattu push-
notifikaatiot.
     * @return Lista käyttäjistä joille tilattu push-notifikaatiot.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/g
etUsers")
```

```
    public User[] getUsers();

    /**
     * Käyttäjän push-notifikaatioiden tilan päivitys.
     * @param user Päivitetyt tiedot käyttäjästä.
     */
    @WebMethod(action =
"http://sofokus.com/schemas/imgr/calendar/2010/04/21/CalendarService/u
pdateUser")
    public void updateUser(
        @WebParam(name = "user")
        User user);

}
```